



**SUN YAT-SEN UNIVERSITY**

中山大學



# 手机应用平台软件开发

---

## 5、事件处理

---



# Android Framework构成

SUN YAT-SEN UNIVERSITY

## □ 四大元素

- Activities-管理应用程序展示
- Services-管理后台服务
- Broadcast receivers
- Content Provider-管理数据共享

## □ 沟通的桥梁: Intent



SUN YAT-SEN UNIVERSITY

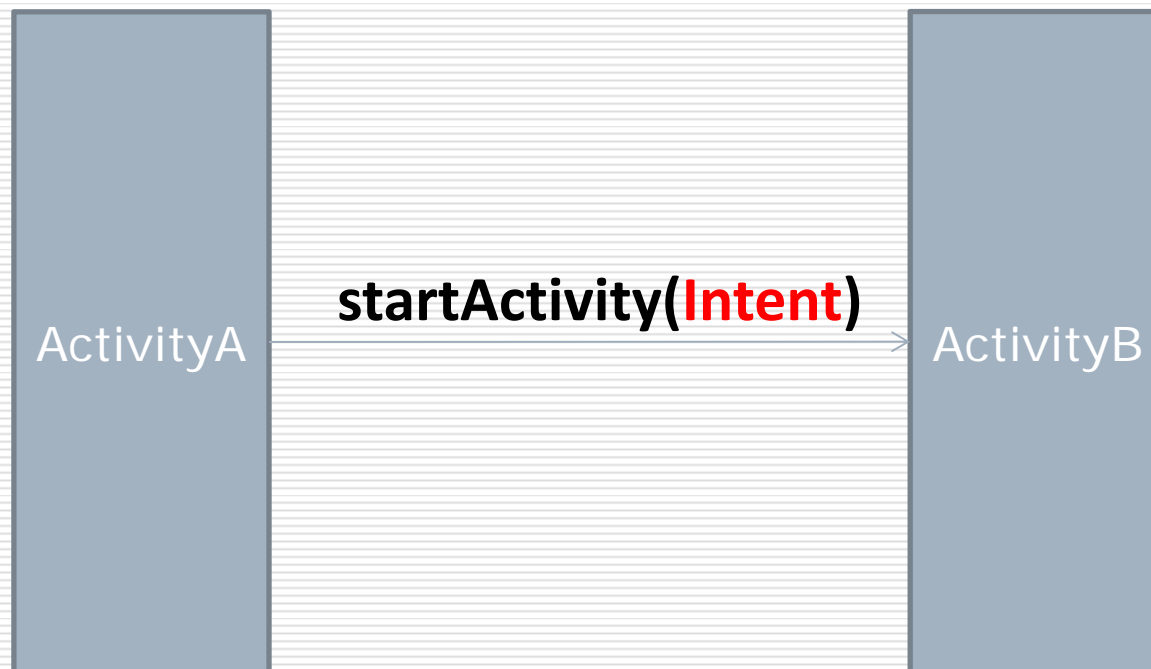


# Activity



# 多个Activity之间的关系

SUN YAT-SEN UNIVERSITY

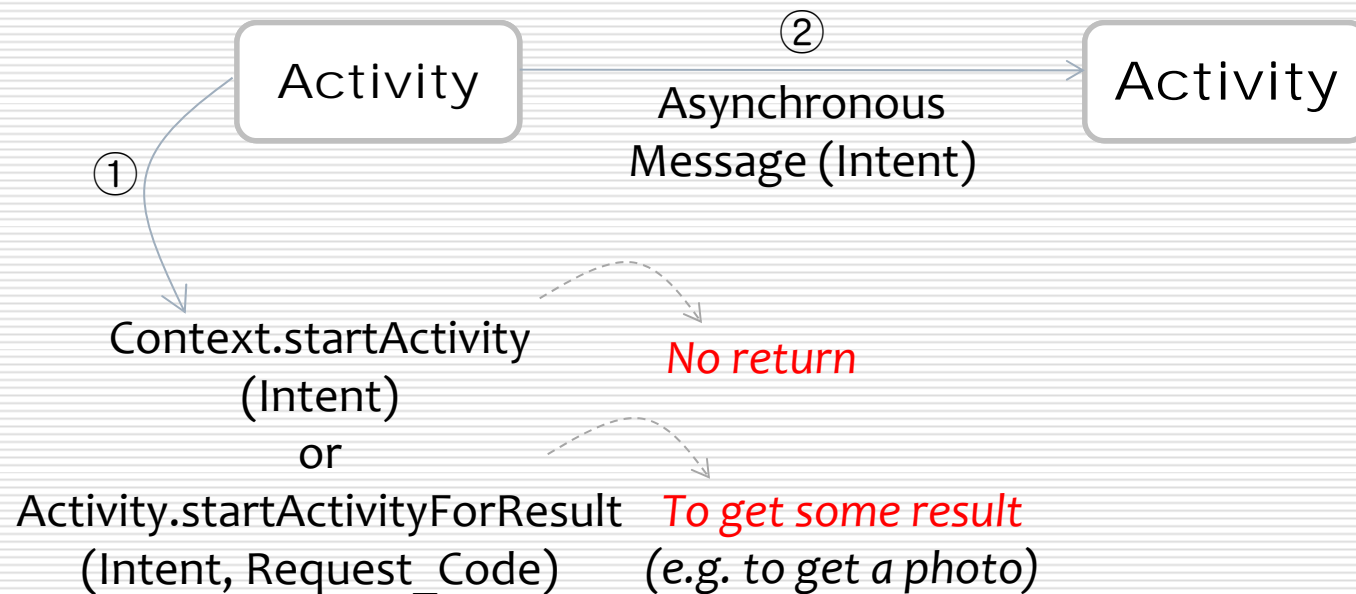




# 活动 (Activities) 和任务 (Tasks)

SUN YAT-SEN UNIVERSITY

- ❑ One activity can start another, including one defined in a different application.





# 启动新的Activity(一)

SUN YAT-SEN UNIVERSITY

通过startActivity(Intent intent)方法打开新的Activity。在打开新的Activity前，  
可以决定是否为新的Activity传递参数。

第一种：打开新的Activity，不传递参数

```
public class MainActivity extends Activity {  
    protected void onCreate(Bundle savedInstanceState) {.....  
        Button button =(Button) this.findViewById(R.id.button);  
        button.setOnClickListener(new View.OnClickListener(){//点击该按钮会打开一个  
新的Activity  
            public void onClick(View v) { //新建一个Intent，第一个参数为当前Activity类  
对象，第二个参数为你要打开的Activity类  
                startActivity(new Intent(MainActivity.this, NewActivity.class));  
            }); }  
    }); }
```



# 启动新的Activity(二)

SUN YAT-SEN UNIVERSITY

## 第二种：打开新的Activity，传递参数

```
public class MainActivity extends Activity {  
    protected void onCreate(Bundle savedInstanceState) {  
        .....  
        button.setOnClickListener(new View.OnClickListener(){//点击该按钮会打开一个新的Activity  
            public void onClick(View v) {  
                Intent intent = new Intent(MainActivity.this, NewActivity.class)  
                Bundle bundle = new Bundle();//该类用作携带数据  
                bundle.putString("name", "tom");  
                bundle.putInt("age", 4);  
                intent.putExtras(bundle);//附上额外的数据  
                startActivity(intent);  
            }  
        });  
    }  
}
```

在新的Activity中接收前面Activity传递过来的参数：

```
public class NewActivity extends Activity {  
    protected void onCreate(Bundle savedInstanceState) {  
        Bundle bundle = this getIntent().getExtras();  
        String name = bundle.getString("name");  
        int age = bundle.getInt("age");    }    }
```



# 获取Activity返回的数据

SUN YAT-SEN UNIVERSITY

若需在Activity中得到新打开Activity 关闭后返回的数据，需使用系统提供的

**startActivityForResult(Intent intent, int requestCode)**方法打开新的

Activity，新的Activity 关闭后会向前面的Activity 传回数据，为了得到传回的数据，

必须在前面的Activity中重写onActivityResult(int requestCode, int resultCode,

Intent data)方法：

---





# 获取Activity返回的数据

SUN YAT-SEN UNIVERSITY

```
public class MainActivity extends Activity {  
    @Override protected void onCreate(Bundle savedInstanceState) { .....  
        Button button = (Button) this.findViewById(R.id.button);  
        button.setOnClickListener(new View.OnClickListener(){  
            public void onClick(View v) { //第二个参数为请求码，可以根据业务需求自己编号  
                startActivityForResult (new Intent(MainActivity.this, NewActivity.class),1); } } );  
        //第一个参数为请求码，即调用startActivityForResult()传递过去的值 第二个参数为结果码  
    @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
        String result = data.getExtras().getString("result");//得到新Activity 关闭后返回的数据 }  
    }
```

当新Activity关闭后，新Activity返回的数据通过Intent进行传递，android平台会调用前面Activity 的onActivityResult()方法，把存放了返回数据的Intent作为第三个输入参数传入，在onActivityResult()方法中使用第三个输入参数可以取出新Activity返回的数据。



# 得到Activity返回的数据

SUN YAT-SEN UNIVERSITY

使用startActivityForResult(Intent intent, int requestCode)方法打开新的Activity，新Activity关闭前需要向前面的Activity返回数据需要使用系统提供的**setResult(int resultCode, Intent data)**方法实现：

```
public class NewActivity extends Activity {  
    @Override protected void onCreate(Bundle savedInstanceState) {  
        .....  
        button.setOnClickListener(new View.OnClickListener(){  
            public void onClick(View v) {  
                Intent intent = new Intent();//数据是使用Intent返回  
                intent.putExtra("result", "hello");//把返回数据存入Intent  
                NewActivity.this.setResult(RESULT_OK, intent);//设置返回数据  
                NewActivity.this.finish();//关闭Activity  
            }  
        });  
    }  
}
```

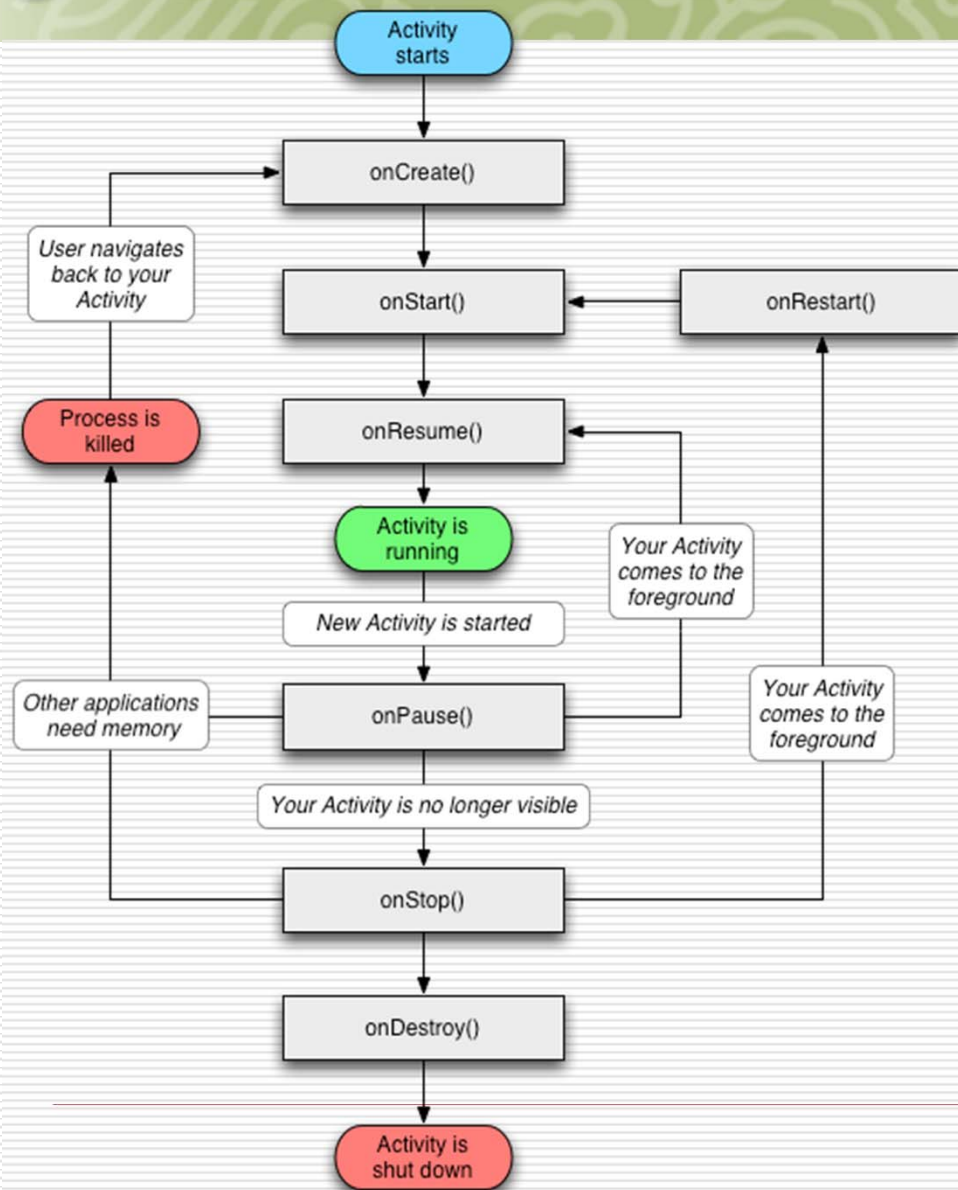
**setResult()**方法的第一个参数值可以根据业务需要自己定义，上面代码中使用到的RESULT\_OK是系统Activity类定义的一个常量，值为-1

---



# Activity生命周期

SUN YAT-SEN UNIVERSITY



重要数据都写在  
`onPause()`里面，最后  
一个安全生命周期方法



# 四种状态

SUN YAT-SEN UNIVERSITY

Android 的虚拟机(VM)是使用基于栈(Stack based) 管理。主要有四种状态:

- ☐ Active (活动)
  - ☐ Paused (暂停)
  - ☐ Stopped (停止)
  - ☐ Dead (已回收或未启动)
-



# Active (活动)

SUN YAT-SEN UNIVERSITY

"Active"状态是使用者启动应用程序或Activity 后, Activity 运行中的状态。在Android 平台上, 同一个时刻只会有一个Activity 处于活动(Active)或运行(Running)状态。其他的Activity 都处于未启动(Dead)、停止(Stopped)、或是暂停(Pause)的状态。

---



# Paused (暂停)

SUN YAT-SEN UNIVERSITY

"Paused"状态是当Activity 暂时暗下来，退到背景画面的状态。

当使用Toast、AlertDialog、或是电话来了时，都会让原本运行的Activity 退到背景画面。新出现的Toast、AlertDialog 等界面元件盖住了原来的Activity 画面。Activity 处在"Paused"状态时，使用者无法与原Activity 互动。

---



# Stopped (停止)

SUN YAT-SEN UNIVERSITY

Stopped“状态是有其他Activity 正在执行，而这个Activity 已经离开屏幕，不再动作的状态。通过长按“Home”钮，可以叫出所有处于“Stopped”状态的应用程序列表。在“Stopped”状态的Activity，还可以通过“Notification”来唤醒。

---



# Dead (已回收或未启动)

SUN YAT-SEN UNIVERSITY

- "Dead"状态是Activity 尚未被启动、已经被手动终止，或已经被系统回收的状态。
  - 要手动终止Activity，可以在程序中调用“finish”函数，若是被系统回收，可能是因为内存不足，因此系统根据内存不足时的回收规则，将处于“Stopped”状态的Activity 所占用的内存回收。
-





# Activity 生命周期

SUN YAT-SEN UNIVERSITY

## ➤ onCreate()

- Called when the activity is first created or when the activity was killed

## ➤ onStart()

- Called just before the activity becomes visible to user

## ➤ onRestart()

- Called after the activity has been stopped, just prior to it being started again
-



# Activity 生命周期

SUN YAT-SEN UNIVERSITY

## ➤ **onResume()**

- Called just before the activity starts interacting with the user
- At this point, the activity is at the top of the activity stack, with user input going to it.

## ➤ **onPause()**

- Called when the system is about to start resuming another activity
  - This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on.
-



# Activity 生命周期

SUN YAT-SEN UNIVERSITY

## ➤ **onStop()**

- Called when the activity is no longer visible to the user
- This may happen because it is being destroyed, or because another activity has been resumed and is covering it.

## ➤ **onDestroy()**

- Called before the activity is destroyed
-



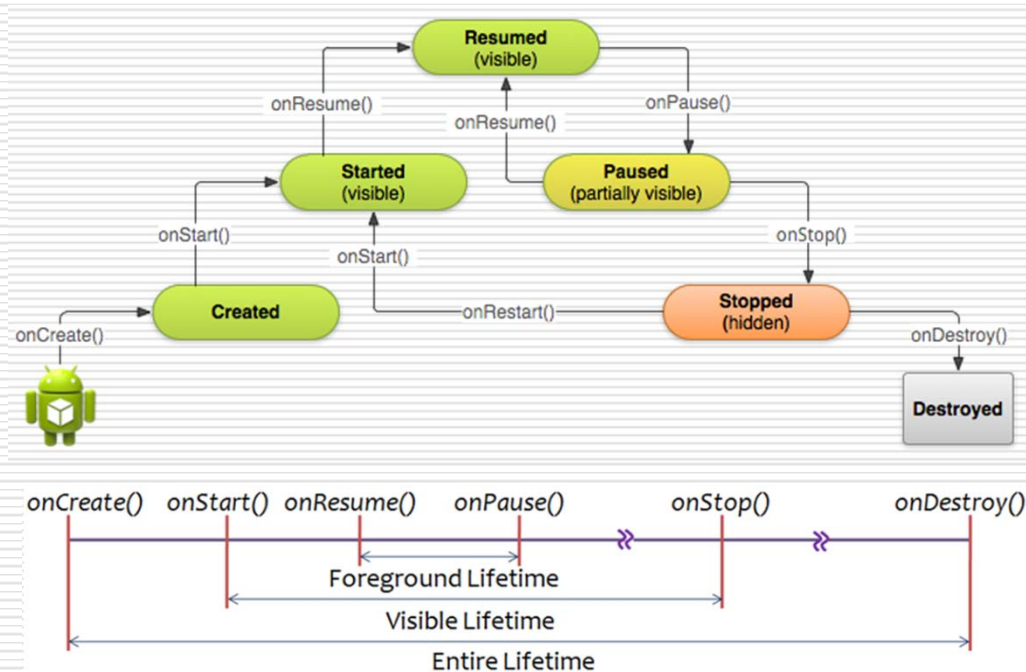
# Activity生命周期分类

SUN YAT-SEN UNIVERSITY

## ➤ 完整生命周期

– onCreate()/ onDestroy()

--完整的Activity 生命周期由"Create"状态开始，由"Destroy"状态结束。建立(Create)时分配资源，销毁(Destroy)时释放资源。





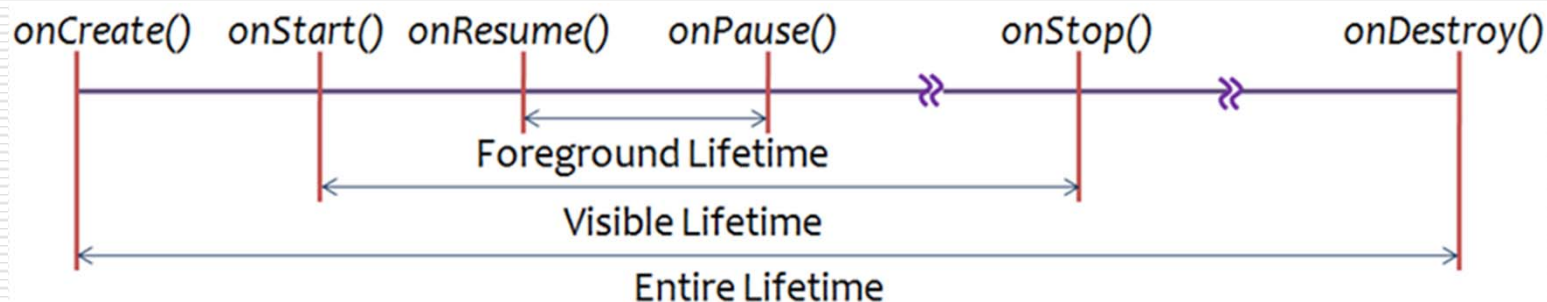
# Activity生命周期分类

SUN YAT-SEN UNIVERSITY

## ➤ 可视生命周期

– onStart()/onStop()

--当Activity 运行到“Start”状态时，就可以在屏幕上看到这个Activity。相反地，当Activity 运行到“Stop”状态时，这个Activity 就会从屏幕上消失。当使用者按下Back 按钮回到上一个Activity 时，会先到Restart 状态，再到一般的Start 状态。





# Activity生命周期分类

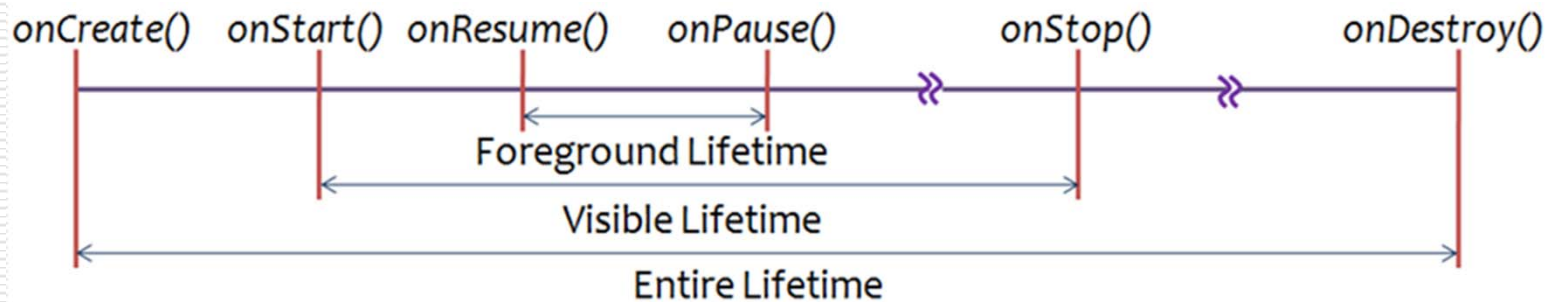
SUN YAT-SEN UNIVERSITY

## ➤ 前台生命周期

– onResume()/onPause()

--使用者能否直接存取萤幕（Resume/Pause）

当有个Toast、AlertDialog、短信、电话等信息发生时，原来的Activity 会处于“Pause”状态，暂时放弃直接存取屏幕的能力，被中断到背景去，将前景交给优先级高的事件。当这些优先级高的事件处理完后，Activity 就改进入"Resume"状态，此时又直接存取屏幕。





# 任务栈的概念

SUN YAT-SEN UNIVERSITY

□ 压栈

□ 弹栈

第三个Activity

第二个Activity

第一个Activity



# 任务栈运行过程

SUN YAT-SEN UNIVERSITY

- 应用程序启动后，运行第一个Activity之后，该Activity对象被压入到Stack之中







# 任务栈运行过程

SUN YAT-SEN UNIVERSITY

点击按钮后启动第二个Activity，该对象被压入到Stack中

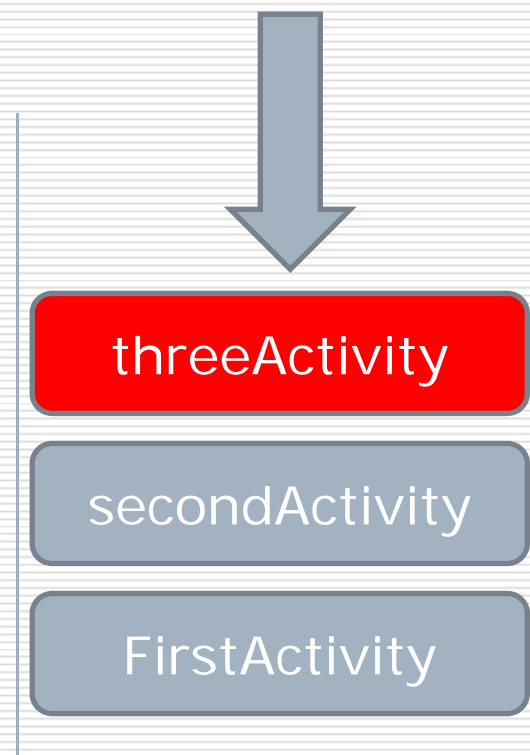




# 任务栈运行过程

SUN YAT-SEN UNIVERSITY

点击第二个Activity按钮启动，该对象被压入到Stack中





# 任务栈运行过程

SUN YAT-SEN UNIVERSITY

- 当点击第三Activity中的按钮启动之后，启动第四个Activity



SMSActivity

threeActivity

secondActivity

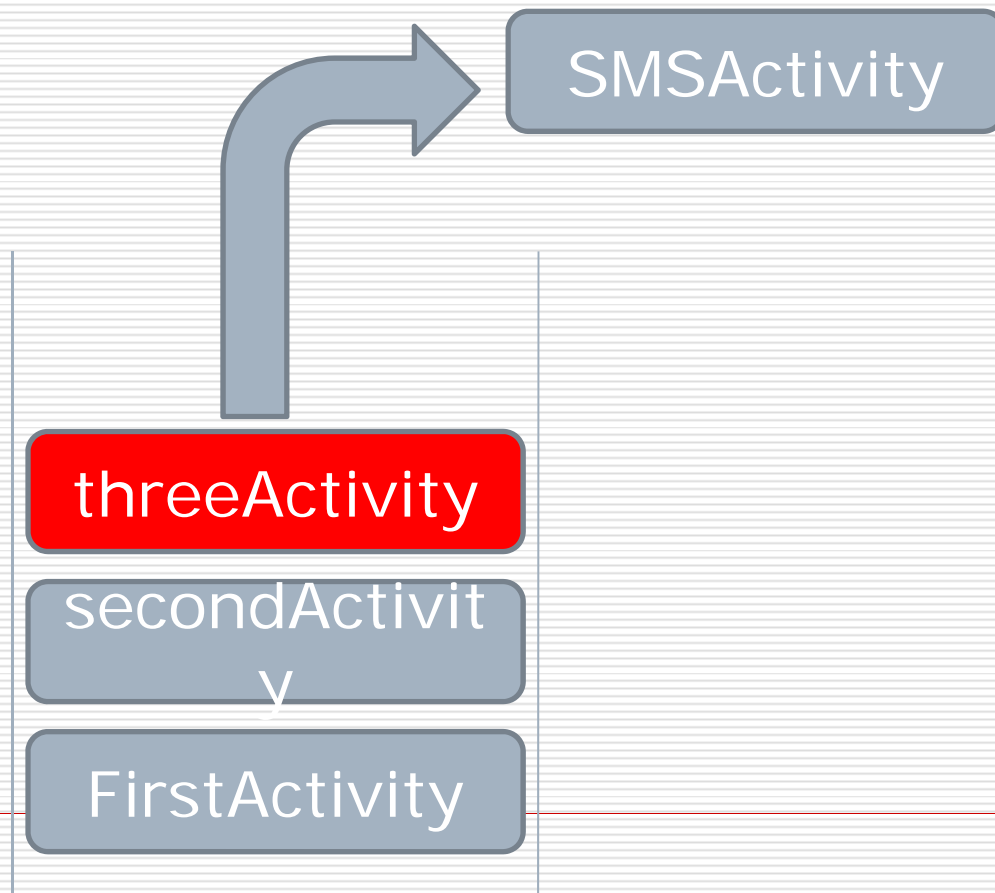
FirstActivity



# 任务栈运行过程

SUN YAT-SEN UNIVERSITY

- ❑ 点击Back之后，这时SMSActivity从堆栈中弹出。
- ❑ 后面的对象都是一样。
- ❑ 所有的Activity在同一个Task，被组织称同一个单元。





# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

一个应用程序通常包含多个**Activity**。每个**Activity**都必须设计成一种特定的操作，用户可以通过该操作去实现某项功能，并且操作其他的**Activity**。

例如：一个电子邮件的应用程序可能有一个**Activity**，用于展现出新的电子邮件列表，当用户选择了一个电子邮件，就打开一个新的**Activity**以查看该电子邮件。

---



# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

当用户进行某项操作时，任务栈就收集相互交互的**Activity**。

**Activity**会被安排在堆栈中(返回堆栈)，堆栈中的**Activity**会按顺序来重新打开。

一个设备的主屏幕是大多数任务栈的起点。当用户触摸图标(或者在屏幕上的快捷方式)时，该应用程序就会到达任务栈的最前面。如果该应用在任务栈中不存在(应用在近段时间内没有使用过)就会在任务栈中创建一个新的任务，并将该应用作为"主"**Activity**放置在根任务栈中。

---



# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

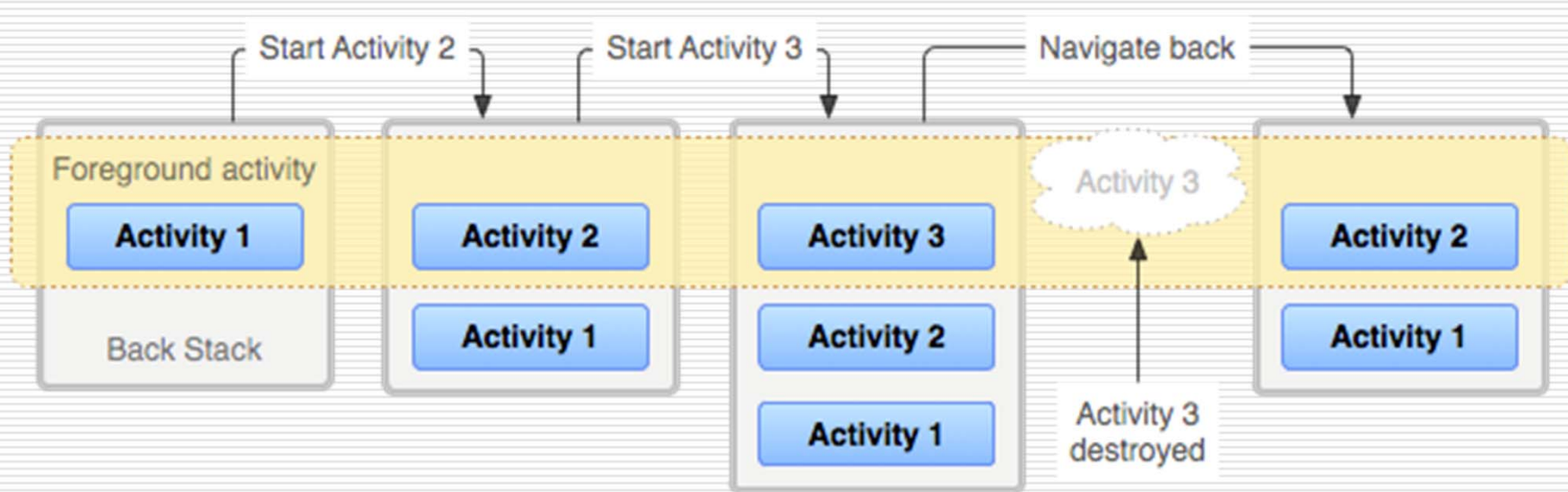
当Activity开始时，新的Activity推入堆栈的顶部和焦点。以前的Activity仍然在堆栈中，但已停止。当Activity停止时，系统会保留其用户界面的当前状态。当用户按下返回按钮时，当前Activity就会从堆栈的顶部(当前的Activity就会被销毁)和之前的一个Activity就会恢复(恢复到之前的UI界面)。Activity在栈中的顺序永远不会改变，只会压入和弹出——被当前Activity启动时压入栈顶，用户用返回键离开时弹出。这样，back stack 以“后进先出”的方式运行。下图以时间线表的方式展示了多个Activity切换时对应当前时间点的Back Task 状态。

---



# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

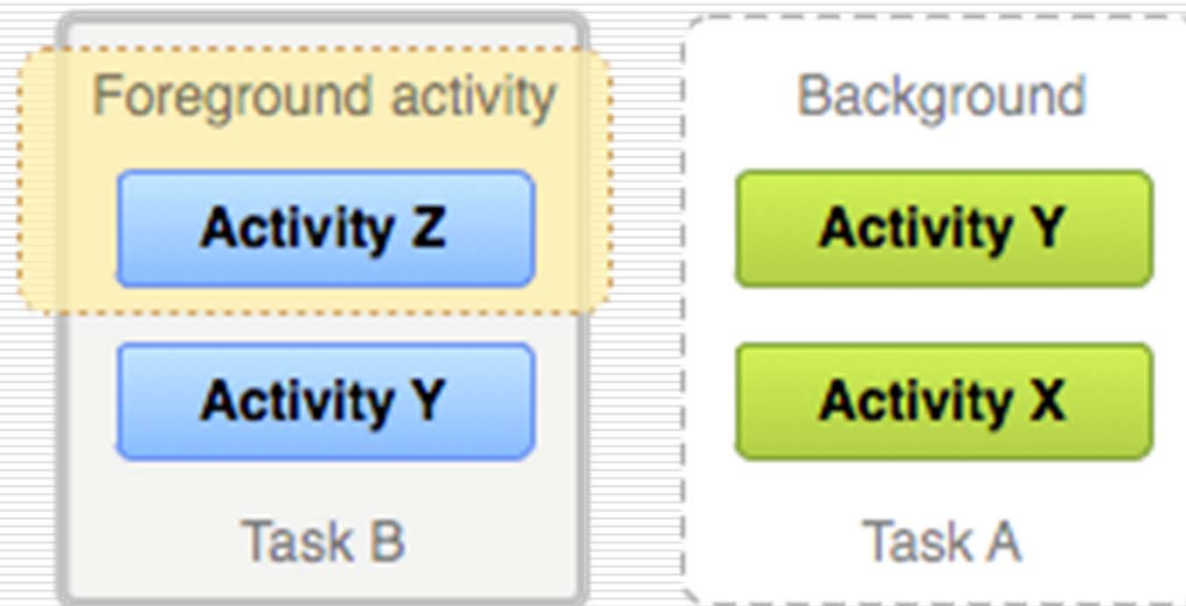






# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

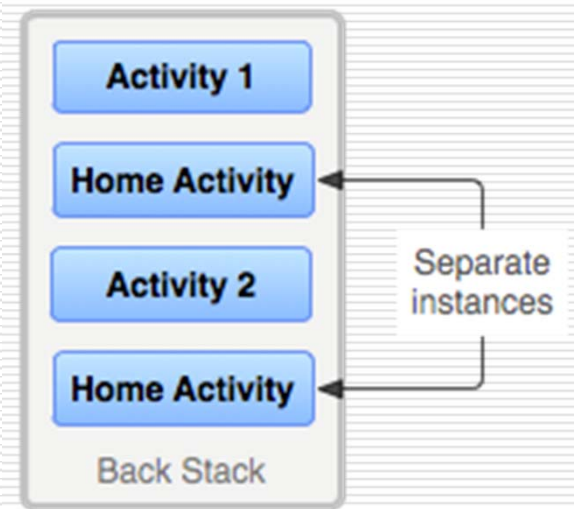




# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

因为Back Stack 中的Activity顺序永远不会改变，如果应用允许某个Activity 可以让用户启动多次，则新的实例会压入栈顶（而不是打开之前位于栈顶的Activity）。于是一个Activity 可能会初始化多次（甚至会位于不同的Task 中），如图所示.如果用户用返回键返回时，Activity 的每个实例都会按照原来打开的顺序显示出来（用户界面也都按原来状态显示.当然，如果你不想让Activity 能被多次实例化，你可以改变它.





# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

## Activity和Task的默认行为

- 当 Activity A 启动 Activity B 时,Activity A 被停止,但系统仍会保存Activity A 状态（比如滚动条位置和 form 中填入的文字）如果用户在 Activity B 中按下返回键时,Activity A 恢复运行,状态也将恢复.
  - 当用户按下Home键离开Task 时,当前 Activity 停止Task 转入后台,系统会保存Task中每个 Activity 的状态。如果用户以后通过选中启动该 Task 的图标来恢复Task,Task 就会回到前台, 栈顶的Activity 会恢复运行.
  - 如果用户按下返回键,当前Activity 从栈中弹出,并被销毁.栈中前一个Activity恢复运行.当 Activity 被销毁时,系统不会保留Activity 的状态.
  - Activity甚至可以在不同的Task中被实例化多次
-



# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

- 系统默认会在**Activity** 停止时保存其状态.这样,当用户返回时,用户的界面能与离开时显示得一样.不过也应该使用回调方法主动地保存**Activity**的状态,以便应对**Activity**被销毁并重建的情况.
  - 当系统停止一个**Activity** 运行后 (比如启动了一个新**Activity** 或者**Task** 转入后台),系统在需要回收内存的时候有可能会完全销毁该**Activity**.这时,该**Activity**的状态信息将会丢失.就算这种情况发生,该**Activity** 仍然会存在于**Back Stack**中.但是当它回到栈顶时, **系统将必须重建它 (而不是恢复)**.为了避免用户工作内容的丢失, 你应通过实现**Activity** 的 **onSaveInstanceState()** 方法来主动保存这些内容.
-



# 任务栈和返回堆栈 (Tasks & Back Stack)

SUN YAT-SEN UNIVERSITY

如上所述,把所有已经启动的**Activity** 相继放入同一个**Task** 中以及一个“后入先出”栈, **Android** 管理**Task**和**Back Stack** 的这种方式适用于大多数应用,一般不用去管理的**Activity** 如何与**Task**关联及如何弹出**Back Stack** .不过,有时候需要改变这种普通的运行方式.也许你想让某个**Activity**启动一个新的**Task** (而不是被放入当前**Task**中), 或者,你想让 **activity** 启动时只是调出已有的某个实例 (而不是在**Back Stack** 顶创建一个新的实例) 或者,你想在用户离开**Task** 时只保留根**Activity**, 而**Back Stack** 中的其它 **Activity** 都要清空,

---



# Intent

SUN YAT-SEN UNIVERSITY

- Intent是一种运行时绑定机制，他能在应用程序运行的过程中连接两个不同的组件，实现组件间的跳转。
  - Activity、Service和BroadcastReceiver，都是通过Intent机制激活的，而不同类型的组件在传递Intent时有不同的方式。
-



# Intent

SUN YAT-SEN UNIVERSITY

- An intent is an abstract description of an operation to be performed: 一个Intent就是一次对将要执行的操作的抽象描述。
  - **Intent** 数据结构两个最重要的部分是:
    - 动作: 典型的动作类型有: MAIN (活动的门户)、VIEW、PICK、EDIT等。
    - 动作对应的数据: 以**URI** 的形式进行表示  
例如: 要查看某个人的联系方式, 你需要创建一个动作类型为VIEW 的Intent, 以及一个表示这个人的URI。
-



- Android 使用了Intent 这个特殊类，实现在屏幕与屏幕之间跳转。Intent 类用于描述一个应用将会做什么事。
- 目前有三种**intent**
  1. 启动一个新的activity，并可以携带数据；
  2. 通过一个intent来启动一个服务；
  3. 通过intent来广播一个事件；





# Intent例程（页面跳转）

SUN YAT-SEN UNIVERSITY

```
□ import android.content.Intent;
```

```
    Intent intent = new Intent(Startup.this, MainList.class);  
    startActivity(intent);
```

Intent显式调用，页面就从Startup.java跳转到MainList.java了。

---



- 每个**intent**都带有一个动作（**action**），并根据不同的动作去行动。

```
public void onClick()  
{ Uri uri = Uri.parse("http://www.sysu.edu.cn/");  
Intent intent = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(intent); }  
Listener = new OnClickListener(){  
Public void onClick(View v)  
{ setTitle("Test Intent");  
Intent int=new Intent(ActivityMain.this,Activity2.class);  
startActivity(intent)
```

- 要开启一个网页，由**Android**去决定谁要开启网页。



# Intent激活方式

SUN YAT-SEN UNIVERSITY

- ❑ **Activity:** 通过调用Context.startActivity()或者Activity.startActivityForResult()方法。
  - ❑ **Service:** 调用Context.startService()或者Context.bindService()方法将调用这方法的上下文与Service绑定。
  - ❑ **BroadcastReceiver:** 通过Context.sendBroadcast()、Context.sendOrderBroadcast()和Context.sendStickyBroadcast()
  - ❑ **Broadcast()**发送BroadcastIntent。BroadcastIntent发送后，所有已注册的拥有与之匹配的IntentFilter的BroadcastIntent就会被激活。
-



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## □ 主要包括6部分：

- 组件名称：Intent目标组件的名称。
  - 动作(Action)：描述Intent所触发动作名字的字符串。
  - Data：描述Intent要操作的数据URI和数据类型。
  - Category：对被请求组件的额外描述信息。
  - Extra：当我们使用Intent连接不同的组件时，有时需要在Intent添加额外的信息，以便把数据传递给目标Activity。
  - Flag：借助 Intent 中的 flag 和 AndroidManifest.xml 中 activity 元素的属性，就可以控制到 Task 里 Activity 的关联关系和行为。
-

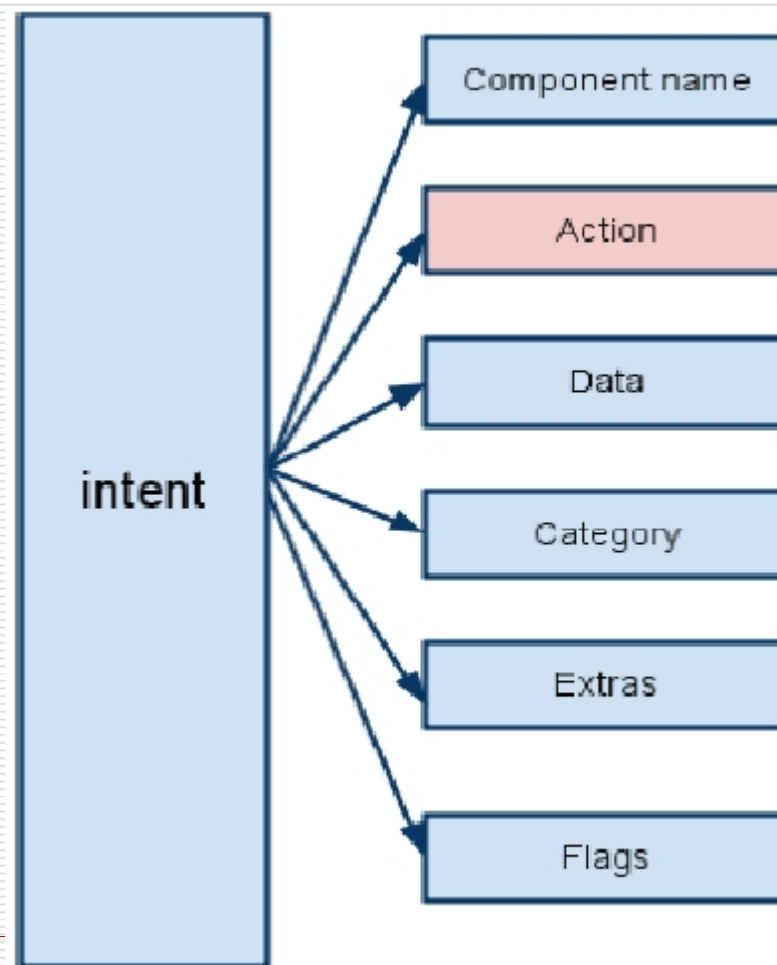


# Intent的组成部分

SUN YAT-SEN UNIVERSITY

一个Intent对象主要包含六类信息。

并非每个都需要包含这六类信息。





# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Component name

- Component属性 明确指定Intent的**目标组件**的类名称
  - 通常 Android框架会根据Intents 中包含的其它属性的信息，比如 action、data/type、category进行查找，最终找到一个匹配的目标组件。但是，如果 Component这个属性有指定的话，将直接使用它指定的组件，而不再执行上述查找过程。指定了这个属性以后，Intents的其它所有属性都是可选的。
-



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Action

- 对执行动作的描述：操作(Action)
  - 系统自定义了很多Action,
  - ACTION\_MAIN ， 我们最熟悉的一个。 “android.intent.action.MAIN” ， 这个值我们在每个AndroidManifest.xml文档中都可以看到。它标记当前Activity作为一个程序的入口。
  - 自己也可以定义自己的Action
-



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Data

- 数据(data): 对于这次动作相关联的数据进行描述, 表现成为一个URI





# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Action 和 Data 示例

### 1. 打开浏览器显示网页

- `Uri uri = Uri.parse("http://www.ataaw.com");`
- `Intent intent = new Intent(Intent.ACTION_VIEW,uri);`
- `startActivintenty(intent);`

### 2. 拨打电话，调用拨号程序：

- `Uri uri = Uri.parse("tel:13800138000");`
  - `Intent intent = new Intent(Intent.ACTION_DIAL, uri);`
  - `startActivintenty(intent);`
-



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Action 和 Data 示例

### 1. 调用短信程序发送短信

- `Uri uri = Uri.parse("smsto:13800138000");`
- `Intent intent = new Intent(Intent.ACTION_SENDTO, uri);`
- `intent.putExtra("sms_body", "ATAAW.COM");`
- `startActivintenty(intent);`

### 2. 从通讯录中选择一个联系人

- `Intent intent = new Intent(Intent.ACTION_PICK,`  
`ContactsContract.Contacts.CONTENT_URI);`
  - `startActivityResult(intent, 0);`
  - 在返回的Intent中getData()可以得到选中的联系人的URI
-



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Category

类别(category):对执行动作的附加信息进行描述, 例如:

- CATEGORY\_DEFAULT
  - CATEGORY\_BROWSABLE 可以由浏览器打开
  - CATEGORY\_TAB 可以嵌入带选项卡的父活动
  - CATEGORY\_LAUNCHER 顶级活动,在启动屏幕上列出该活动
  - CATEGORY\_HOME 启动手机时或按下Home键时运行
-



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## CATEGORY\_HOME 示例

```
<activity android:name=".HelloPicActovity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.HOME" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

按下HOME键时,该应用会被运行

---



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Extras

- 附件信息(extras)，是用来传递数据的。
  - 它是一个Bundle类的对象，有一组可序列化的key/value 对组成。
-



# Intent的组成部分

SUN YAT-SEN UNIVERSITY

## Flags

- 标记(flag):指导如何来启动一个Activity，是用来指明运行的模式。
  - 例如，期望这个Intent的执行者和运行在两个完全不同的任务进程中，就需要设置FLAG\_ACTIVITY\_NEW\_TASK 的标志位。
-



# Activity间数据传递

SUN YAT-SEN UNIVERSITY

```
public void onClick(View v) {  
    Intent intent1 = new Intent(ActivityMain.this, Activity1.class);  
    intent1.putExtra("activityMain", "数据来自activityMain");  
    startActivityForResult(intent1, REQUEST_CODE);  
  
    String data=null;  
    Bundle extras = getIntent().getExtras();  
    if (extras != null) {  
        data = extras.getString("activityMain");  
    }  
    setTitle("现在是在Activity1里:"+data);  
}
```



# Bundle类型

SUN YAT-SEN UNIVERSITY

Bundle是个类型安全的容器，保存的是名值对，值只能是基本类型或基本类型数组。如：String、Int、byte、boolean、char等。





# 为Intent附加数据

SUN YAT-SEN UNIVERSITY

Intent提供了各种常用类型重载后的putExtra()方法，如： putExtra(String name, String value)、 putExtra(String name, long value)，在putExtra()方法内部会判断当前Intent对象内部是否已经存在一个Bundle对象，如果不存在就会新建Bundle对象，以后调用putExtra()方法传入的值都会存放于该Bundle对象，下面是Intent的putExtra(String name, String value)方法代码片断：

```
public class Intent implements Parcelable {  
    private Bundle mExtras;  
    public Intent putExtra(String name, String value) {  
        if (mExtras == null) {  
            mExtras = new Bundle();  
        } mExtras.putString(name, value);  
    }  
    return this; }  
}
```

---



# 为Intent附加数据

SUN YAT-SEN UNIVERSITY

第一种写法：用于批量添加数据到Intent：

```
Intent intent = new Intent();  
  
Bundle bundle = new Bundle();  
  
bundle.putString("name", "tom");  
  
intent.putExtras(bundle);
```

第二种写法：等价于上面的写法，只不过这种写法是把数据一个个地添加进Intent，这种写法使用起来比较方便，而且只需要编写少量的代码。

```
Intent intent = new Intent();  
  
intent.putExtra("name", "tom");
```

---



# IntentFilter

SUN YAT-SEN UNIVERSITY

- **intentfilter** 则用于描述一个**activity**（或**IntentReceiver**）能够操作哪些**intent**。
    - 一个**activity** 如果要显示一个人的联系方式时，需要声明一个**IntentFilter**，这个**IntentFilter** 要知道怎么去处理**VIEW** 动作和表示一个人的**URI**。
  - **IntentFilter** 需要在**AndroidManifest.xml** 中定义。
-



# IntentFilter

SUN YAT-SEN UNIVERSITY

- ❑ **Intent Filter** : 向系统说明该**Activity**可以做什么？
  - ❑ 通过解析各种**intent**，从一个屏幕导航到另一个屏幕是很简单的。
    - 当向前导航时，**activity** 将会调用**startActivity(Intent myIntent)**方法。
    - 然后，系统会在**所有安装的应用程序中定义的IntentFilter** 中查找，找到最匹配**Intent** 对应的**activity**。
    - 新的**activity** 接收到通知后，开始运行。
    - 当**startActivity** 方法被调用将触发解析**Intent** 的动作。
  - ❑ 该机制提供了两个关键好处：
    - **Activities** 能够重复利用从其它组件中以**Intent** 的形式产生的一个请求；
    - **Activities** 可以在任何时候被一个具有相同**IntentFilter** 的新的**Activity** 取代。
-



# IntentReceiver

SUN YAT-SEN UNIVERSITY

- ❑ IntentReceiver 在AndroidManifest.xml 中注册，也可在代码中使用 Context.registerReceiver()进行注册。
  - ❑ 当一个intentreceiver 被触发时，应用不必对请求调用intentreceiver，系统会在需要的时候启动你的应用。
  - ❑ 各种应用还可以通过使用Context.broadcastIntent()将自己的 intentreceiver 广播给其它应用程序。
  - ❑ 当希望应用能够对一个外部的事件(如电话呼入，数据网络可用，或者某个定时)做出响应，可以使用一个IntentReceiver。虽然IntentReceiver 在感兴趣的事件发生时，会使用NotificationManager通知用户，但它并不能生成一个UI。
-



# Intent的调用方式

SUN YAT-SEN UNIVERSITY

## Intent分为显式和隐式两种：

- **显式Intent**：明确指出了目标组件名称。显式Intent更多用于在应用程序内部传递消息。比如在某应用程序内，一个Activity启动一个Service。
  - **隐式Intent**：没有明确指出了目标组件名称。Android系统使用IntentFilter 来寻找与隐式Intent相关的对象。隐式Intent恰恰相反，它不会用组件名称定义需要激活的目标组件，它更广泛地用于在不同应用程序之间传递消息。
-



# 自定义Action

SUN YAT-SEN UNIVERSITY

- ❑ Explicit Intent明确的指定了要启动的Activity，比如以下Java代码：`Intent intent = new Intent(this, B.class)`
- ❑ Implicit Intent没有明确的指定要启动哪个Activity，而是通过设置一些Intent Filter来让系统去筛选合适的Activity去启动。
- ❑ intent到底发给哪个activity，需要进行三个匹配，一个是action，一个是category，一个是data。
- ❑ 理论上来说，如果intent不指定category，那么无论intent filter的内容是什么都应该是匹配的。但是，如果是implicit intent，android默认给加上一个CATEGORY\_DEFAULT，这样的话如果intent filter中没有android.intent.category.DEFAULT这个category的话，匹配测试就会失败。所以，如果你的activity支持接收implicit intent的话就一定要在intent filter中加入android.intent.category.DEFAULT。
- ❑ 例外情况是：android.intent.category.MAIN和android.intent.category.LAUNCHER的filter中没有必要加入android.intent.category.DEFAULT，当然加入也没有问题。
- ❑ 自定义的activity如果接受implicit intent的话，intent filter就一定要加上android.intent.category.DEFAULT这个category。



# 拨打电话/发短信 (1)

SUN YAT-SEN UNIVERSITY

```
Intent intent=new
```

```
Intent(Intent.ACTION_CALL,Uri.parse("tel:"+telPhone));
```

```
Intent intent=new
```

```
Intent(Intent.ACTION_SENDTO,Uri.parse("smsto:5554"));
```

```
intent.putExtra("sms_body", "Hello");
```

```
startActivity(intent);
```

---





## 拨打电话/发短信 (2)

SUN YAT-SEN UNIVERSITY

实际无法拨打电话，安全性问题

```
<uses-permission
```

```
    android:name="android.permission.CALL_PHONE"/>
```

```
<uses-permission
```

```
    android:name="android.permission.SEND_SMS"/>
```

---



# Android事件处理模型

SUN YAT-SEN UNIVERSITY

## ➤ 基于回调的事件处理

- 重写Android等组件的特定的回调方法。在Activity和View类中,定义了若干回调方法,可以通过覆盖这些方法来实现事件处理。

## ➤ 基于监听接口的事件处理

- 为Android界面组件绑定特定的事件监听器

## ➤ Handler

---



# 基于回调的事件处理

SUN YAT-SEN UNIVERSITY

- 在**Activity**和**View**类中,定义了若干回调方法,可以覆盖这些方法来实现事件处理
  - 返回**boolean**值,**true**表示完成了事件处理
    - `onKeyDown(int keyCode,KeyEvent event)`
    - `onKeyUp(int keyCode,KeyEvent event)`
    - `onTouchEvent(MotionEvent event)`
      - `event.getAction() == MotionEvent.ACTION_DOWN`
      - `event.getAction() == MotionEvent.ACTION_UP`
      - `event.getAction() == MotionEvent.ACTION_MOVE`
-



# 基于监听接口的事件处理

SUN YAT-SEN UNIVERSITY

1. 对View调用 setOnXXXListener(listener)

2. View类中若干内部接口:

- OnClickListener : 点击事件
  - OnLongClickListener: 长时点击
  - OnFocusChangeListener: 焦点改变
  - OnKeyListener 键盘事件
  - onTouchListener 触摸事件
  - OnCreateContextMenuListener 上下文菜单显示事件
-



# 基于监听接口的事件处理

SUN YAT-SEN UNIVERSITY

## 1. 事件源Event Source:

产生事件的来源，通常是各种组件，如按钮，窗口等。

## 2. 事件Event:

事件封装了界面组件上发生的特定事件的具体信息，如果监听器需要获取界面组件上所发生事件的相关信息，一般通过事件Event对象来传递。

## 3. 事件监听器Event Listener:

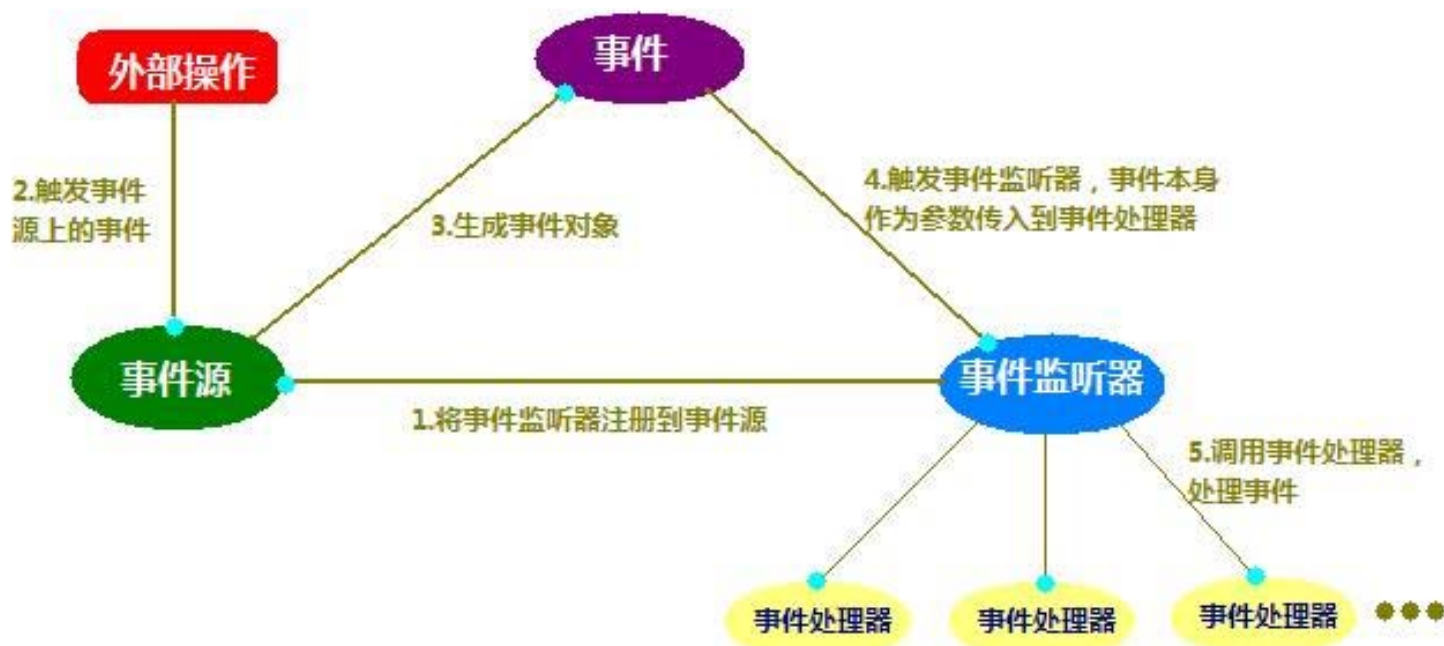
负责监听事件源发生的事件，并对不同的事件做相应的处理。

---



# 基于监听接口的事件处理

SUN YAT-SEN UNIVERSITY



基于监听器的事件处理机制是一种委派式Delegation的事件处理方式，事件源将整个事件委托给事件监听器，由监听器对事件进行响应处理。这种处理方式将事件源和事件监听器分离，有利于提供程序的可维护性。



# OnClickListener

SUN YAT-SEN UNIVERSITY

- ❑ **OnClickListener**接口处理的是点击事件。在触发模式下，是在某个**View** 上按下并抬起的组合动作，而在键盘模式下，是某个 **View** 获得焦点后点击确定键或者按下轨迹事件。
  - ❑ 该接口对应回调方法签名：
    - `Public void onClick (View V)` ; 说明：参数 V 便为事件发生的事件源
-



# OnClickListener例程-(1)

SUN YAT-SEN UNIVERSITY

```
public class Sample extends Activity implements OnClickListener{
//所有事件监听器方法都必须实现事件接口
Button[] buttons = new Button[4];
TextView textView;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        buttons[0] = (Button) this.findViewById(R.id.button01);
        buttons[1] = (Button) this.findViewById(R.id.button02);
        buttons[2] = (Button) this.findViewById(R.id.button03);
        buttons[3] = (Button) this.findViewById(R.id.button04);
        textView = (TextView) this.findViewById(R.id.textView01);
        textView.setTextSize(18);
    }
}
```

---





# OnClickListener例程-(2)

SUN YAT-SEN UNIVERSITY

```
for(Button button : buttons){  
    button.setOnClickListener(this); //给每一个按钮注册监听器  
}  
  
}  
  
@Override  
public void onClick(View v) { //实现事件监听方法  
    if(v == buttons[0]){ //按下第一个按钮时  
        textView.setText("您按下了"+((Button)v).getText()+"，此时是分开处理的！");  
    }  
    else{ //按下其他按钮时  
        textView.setText("您按下了" + ((Button)v).getText());  
    }  
}  
}
```

---



# OnLongClickListener

SUN YAT-SEN UNIVERSITY

- ❑ **OnLongClickListener** 接口和 **OnClickListener** 接口基本原理是相同的，只是该接口是长按事件的捕捉接口，即当长时间按下某个 **View** 时触发的事件。
- ❑ 该接口对应方法的签名为：

**Public boolean onLongClick (View V) ;**

说明：

- 参数 V：参数 V 为事件源控件，当长时间按下此控件时才会触发该方法
  - 返回值：该方法的返回值为一个 **boolean** 类型的变量，当返回为 **true** 时，表示已经完整地处理了这个事件，并不希望其他的回调方法再次进行处理；当返回为 **false** 时，表示并没有完成处理该事件，更希望其他方法继续对其进行处理。
-



# OnLongClickListener例程

SUN YAT-SEN UNIVERSITY

```
public class Sample_7_5 extends Activity implements OnLongClickListener{  
    //实现的接口  
  
    Button button; //声明按钮的引用  
  
    public void onCreate(Bundle savedInstanceState) { //重写的onCreate方法  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        button = (Button) this.findViewById(R.id.button); //得到按钮的引用  
        button.setTextSize(20);  
        button.setOnLongClickListener(this); //注册监听    }  
  
    @Override  
    public boolean onLongClick(View v) { //实现接口中的方法  
        if(v == button){ //当按下的是按钮时  
            Toast.makeText(  
                this,  
                "长时间按下了按钮",  
                Toast.LENGTH_SHORT  
            ).show(); //显示提示  
        }  
        return false;  
    }  
}
```



# OnFocusChangeListener

SUN YAT-SEN UNIVERSITY

□ **OnFocusChangeListener** 接口用来处理控件**焦点发生改变的事件**。如果注册了该接口每当某个控件失去焦点或者获得焦点时都会触发该接口中的回调方法。

□ 该接口对应的签名方法为：

**Public void onFocusChangeListener (View V, Boolean hasFocus)**

■ 说明：

■ 参数 V：参数 V 便为触发该事件的事件源

■ 参数 hasFocus：参数 hasFocus 表示 v 的新状态，即 v 是否获得焦点。

---



# OnFocusChangeListener例程-(1)

SUN YAT-SEN UNIVERSITY

```
public class TestActivity extends Activity implements OnFocusChangeListener{
    TextView myTextView;
    ImageButton[] imageButtons = new ImageButton[4];

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        myTextView = (TextView) this.findViewById(R.id.myTextView);
        imageButtons[0] = (ImageButton) this.findViewById(R.id.button01);
        imageButtons[1] = (ImageButton) this.findViewById(R.id.button02);
        imageButtons[2] = (ImageButton) this.findViewById(R.id.button03);
        imageButtons[3] = (ImageButton) this.findViewById(R.id.button04);

        for(ImageButton imageButton : imageButtons){
            imageButton.setOnFocusChangeListener(this);           //添加监听
        }
    }
}
```

---



# OnFocusChangeListener例程-(2)

SUN YAT-SEN UNIVERSITY

```
public void onFocusChange(View v, boolean hasFocus) {  
    //实现了接口中的方法  
    if(v.getId() == R.id.button01){//改变的是button01时  
        myTextView.setText("您选中了羊!");  
    }else if(v.getId() == R.id.button02){//改变的是button02时  
        myTextView.setText("您选中了猪!");  
    }else if(v.getId() == R.id.button03){//改变的是button03时  
        myTextView.setText("您选中了牛!");  
    }else if(v.getId() == R.id.button04){//改变的是button04时  
        myTextView.setText("您选中了鼠!");  
    }else{//其他情况  
        myTextView.setText("");  
    }  
}  
}
```

---



- OnKeyListener 是对手机键盘进行监听的接口，通过对某个 View 注册该回调方法，当 View 获得焦点并有键盘事件时，便会触发该接口中的回调方法。该接口中的回调方法如下：

```
public Boolean onkey (View v, int keyCode, KeyEvent event) ;
```

说明：

- 参数 v：参数 v 为时间按的事件源控件。
  - 参数 keyCode：参数 keyCode 为手机键盘的键码。
  - 参数 event：参数 event 为键盘事件封装类的对象，其中包含了事件的详细信息
-



# OnKeyListener例程-(1)

SUN YAT-SEN UNIVERSITY

```
public class Sample_7_7 extends Activity implements
    OnKeyListener,OnClickListener{
ImageButton[] imageButtons = new ImageButton[4];//声明按钮数组
TextView myTextView;//声明TextView的引用
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this setContentView(R.layout.main);
    myTextView = (TextView) this.findViewById(R.id.myTextView);
    imageButtons[0] = (ImageButton) this.findViewById(R.id.button01);
    imageButtons[1] = (ImageButton) this.findViewById(R.id.button02);
    imageButtons[2] = (ImageButton) this.findViewById(R.id.button03);
    imageButtons[3] = (ImageButton) this.findViewById(R.id.button04);
    for(ImageButton imageButton : imageButtons){
        imageButton.setOnClickListener(this);//添加单击监听
        imageButton.setOnKeyListener(this);//添加键盘监听
    }
}
```





# OnKeyListener例程(2)

SUN YAT-SEN UNIVERSITY

```
@Override
```

```
public void onClick(View v) { //实现了接口中的方法
    if(v.getId() == R.id.button01){ //改变的是button01时
        myTextView.setText("您点击了按钮A! ");
    }else if(v.getId() == R.id.button02){ //改变的是button02时
        myTextView.setText("您点击了按钮B! ");
    }else if(v.getId() == R.id.button03){ //改变的是button03时
        myTextView.setText("您点击了按钮C! ");
    }else if(v.getId() == R.id.button04){ //改变的是button04时
        myTextView.setText("您点击了按钮D! ");
    }else{ //其他情况
        myTextView.setText("");
    }
}
```

---



# OnKeyListener例程-(3)

SUN YAT-SEN UNIVERSITY

```
@Override
```

```
public boolean onKeyDown(View v, int keyCode, KeyEvent event) { //键盘监听
```

```
switch(keyCode){ //判断键盘码
```

```
case 29: //按键A
```

```
    imageButtons[0].performClick(); //模拟单击
```

```
    imageButtons[0].requestFocus(); //尝试使之获得焦点
```

```
    break;
```

```
case 30: //按键B
```

```
    imageButtons[1].performClick(); //模拟单击
```

```
    imageButtons[1].requestFocus(); //尝试使之获得焦点
```

```
    break;
```

```
case 31: //按键C
```

```
    imageButtons[2].performClick(); //模拟单击
```

```
    imageButtons[2].requestFocus(); //尝试使之获得焦点
```

```
    break;
```

```
case 32: //按键D
```

```
    imageButtons[3].performClick(); //模拟单击
```

```
    imageButtons[3].requestFocus(); //尝试使之获得焦点
```

```
    break;}
```

```
return false;
```

```
}
```

```
}
```



- ❑ **OnTouchListener** 接口是用来处理手机屏幕事件的监听接口，当为 **View** 的范围内触摸按下、抬起或滑下等动作时都会触发该事件。该接口中的监听方法签名如下：

**Public boolean onTouch (View V ,MotionEvent event) ;**

说明：

- 参数 v:： 参数 v 为事件源对象
  - 参数 event: 参数 event 为事件封装类的对象，其中封装了触发事件的详细信息，包括事件的类型、触发时间等信息。
-



# OnTouchListener例程-(1)

SUN YAT-SEN UNIVERSITY

```
public class Sample_7_8 extends Activity { //不用实现接口
    final static int WRAP_CONTENT=-2; //表示WRAP_CONTENT的常量
    final static int X_MODIFY=4; //在非全屏模式下x坐标的修正值
    final static int Y_MODIFY=52; //在非全屏模式下y坐标的修正值
    int xSpan; //在触控笔点击按钮的情况下相对于按钮自己坐标系的
    int ySpan; //x,y位置

    public void onCreate(Bundle savedInstanceState) { //重写的onCreate方法

        super.onCreate(savedInstanceState);

        setContentView(R.layout.main); //设置当前的用户界面

        Button bok=(Button)this.findViewById(R.id.Button01);

        bok.setOnTouchListener( new OnTouchListener(){ //直接把注册
            //事件源，所以不用实现接口并且也不用另外写事件。如果要另外写一个方法，要实现接口并且实现的接口方法要在 public void onCreate () 方法外。
        });
    }
}
```



# OnTouchListener例程-(2)

SUN YAT-SEN UNIVERSITY

```
public boolean onTouch(View view, MotionEvent event) {  
    switch(event.getAction()){  
        case MotionEvent.ACTION_DOWN://触控笔按下  
            xSpan=(int)event.getX();ySpan=(int)event.getY();  
            break;  
        case MotionEvent.ACTION_MOVE://触控笔移动  
            Button bok=(Button)findViewById(R.id.Button01);  
            //让按钮随着触控笔的移动一起移动  
            ViewGroup.LayoutParams lp=  
                new AbsoluteLayout.LayoutParams(//初始化为一个子位置  
                    WRAP_CONTENT, WRAP_CONTENT, (int)event.getRawX()-xSpan-X_MODIFY,  
                    (int)event.getRawY()-ySpan-Y_MODIFY) ;  
            bok.setLayoutParams(lp); break; }  
    return true;  
}}); }
```

---



# OnTouchListener例程-(3)

SUN YAT-SEN UNIVERSITY

```
public boolean onKeyDown (int keyCode, KeyEvent event){//键盘键按下的方法
    Button bok=(Button)this.findViewById(R.id.Button01);
    bok.setText(keyCode+" Down");
    return true;}

public boolean onKeyUp (int keyCode, KeyEvent event){//键盘键抬起的方法
    Button bok=(Button)this.findViewById(R.id.Button01);
    bok.setText(keyCode+" Up");
    return true;}

public boolean onTouchEvent (MotionEvent event){//让按钮随着触控笔的移动一起移动
    Button bok=(Button)this.findViewById(R.id.Button01);
    ViewGroup.LayoutParams lp=new AbsoluteLayout.LayoutParams(
WRAP_CONTENT, WRAP_CONTENT,
    (int)event.getRawX()-xSpan-X_MODIFY, (int)event.getRawY()-ySpan-Y_MODIFY
    ) ;
    bok.setLayoutParams(lp);
    return true;}}
```

---



在**android.os**包中，用于线程间传递消息，线程间需要共享同一个**Handler**对象

1. `handlerMessage(Message msg)`处理消息,需要在接收消息的线程中覆盖该方法
  2. `sendEmptyMessage(int what)`发送空消息,在**handlerMessage**中,通过**msg.what**查看**what**值
  3. `sendMessage(Message msg)` 发送消息
-



# Handler的定义

SUN YAT-SEN UNIVERSITY

## 1. UI控件更新？

- 主线程管理UI控件；
- 耗时操作放在线程中，避免“假死”，导致FC；
- Android主线程非线程安全
- 主线程的UI和子线程间通过Handler；

## 2. Handler主要接受子线程发送的数据，并用此数据配合主线程更新UI.

---





# Handler

SUN YAT-SEN UNIVERSITY

handler可以分发Message对象和Runnable对象到主线程中, 每个Handler实例,都会绑定到创建他的线程中(一般是位于主线程), 它有两个作用: (1): 安排消息或Runnable 在某个主线程中某个地方执行, (2)安排一个动作在不同的线程中执行

Handler中分发消息的一些方法

`post(Runnable)`

`postAtTime(Runnable,long)`

`postDelayed(Runnable long)`

`sendEmptyMessage(int)`

`sendMessage(Message)`

`sendMessageAtTime(Message,long)`

`sendMessageDelayed(Message,long)`

以上post类方法允许你排列一个Runnable对象到主线程队列中, sendMessage类方法, 允许你安排一个带数据的Message对象到队列中, 等待更新.

---

# Questions?



ANDROID