#5 事件处理

刘宁

Android Framework构成

- 主要四部分
  - Activities-管理应用程序展示
    - Activity Life Cycle
  - Services-管理后台服务
    - Services Life Cycle
  - Broadcast receivers
    - 管理事件的广播与接收
  - Content Provider-管理数据共享
    - Android系统中，数据是私有的，包含文件数据和数据库数据。Content Provider类提供了标准方法接口，让其他的应用程序读取。
  - 必须在AndroidManifest.xml中声明
- 沟通的桥梁: Intent

Intent

- 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

- Intent 是描述应用想要做什么（官方表述：Intent是一次即将操作的抽象描述）, 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：要查看某个人的联系方式，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。

Intent

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

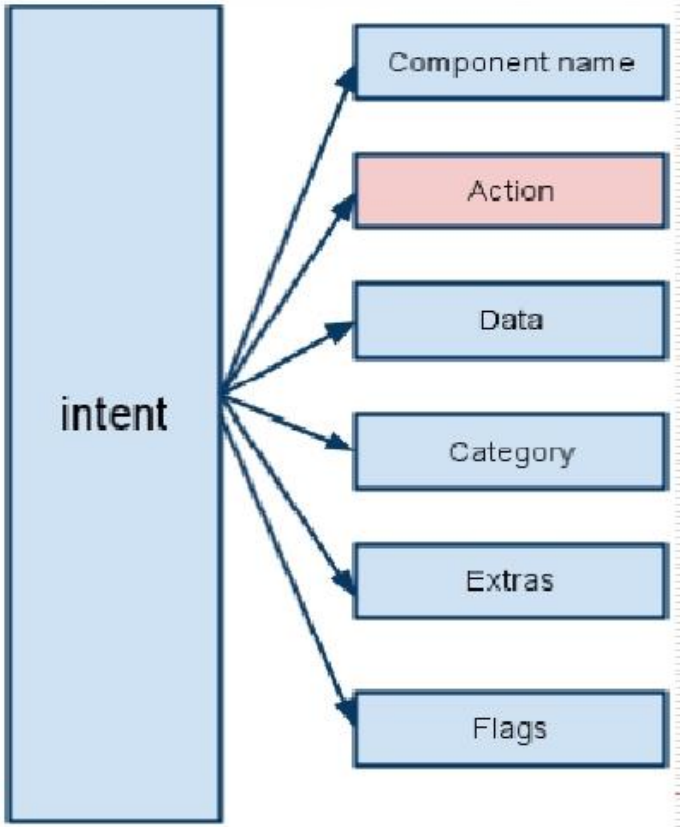
Intent激活方式

- 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.sendStick
- 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

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

- 细分为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 AndroidMainifest.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。

IntentFilter

- IntentFilter 用于描述一个activity（或者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 可在AndroidManifest.xml 中静态定义。

IntentFilter

- 通过解析各种intent，可从一个屏幕导航到另一个屏幕
  - 当向前导航时，activity 可调用startActivity(Intent myIntent)方法。
  - 然后，系统会在所有安装的应用程序定义的IntentFilter 中查找与该Intent最匹配的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。

- 这个机制提供了两个关键好处：
  - Activities 能够重复利用一个Intent 形式的请求；
  - Activities 可以在任何时候被一个具有相同IntentFilter的新的Activity 取代。

IntentReceiver

- 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

- 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。

显式Intent例子

- ```
  import android.content.Intent;
  ```

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

Intent显式调用，页面从Startup Activity跳转到MainList Activity了。
```

隐式Intent例子

- 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.nccu.com.tw/");
  Intent intent = new Intent(Intent.ACTION_VIEW, uri);
  startActivity(intent);
 }
```

- 隐式调用：要开启一个网页，由Android去决定谁要开启网页。

Activity间数据传递

```
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);
```

发送参数

```
String data=null;
Bundle extras = getIntent().getExtras();
  if (extras != null) {
     data = extras.getString("activityMain");
  }
setTitle("现在是在Activity1里:"+data);
```

接收参数

Activity间数据传递—回调

1. 两个Activity: ActivityMain和Activity1
2. 从ActivityMain使用startActivityForResult()打开Activity1，同时传递参数

```
Intent intent1 = new Intent(ActivityMain.this,
Activity1.class);

intent1.putExtra("activityMain", "数据来自activityMain");

startActivityForResult(intent1, REQUEST_CODE);
```

Activity间数据传递—回调

3．Activity1接收到参数，通过setResult()返回参数

```
Bundle bundle = new Bundle();
bundle.putString("store", "数据来自Activity1");
Intent mIntent = new Intent();
mIntent.putExtras(bundle);
setResult(RESULT_OK, mIntent);
finish();
```

Activity间数据传递—回调

4．ActivityMain得到返回结果，在onActivityResult()里处理

```
onActivityResult(int requestCode, int resultCode,
Intent data) {
  if (requestCode == REQUEST_CODE) {
    if (resultCode == RESULT_CANCELED)
      setTitle("取消");
    else if (resultCode == RESULT_OK) {
      String temp=null;
      Bundle extras = data.getExtras();
      if (extras != null) {
        temp = extras.getString("store");
      }
      setTitle(temp);
  }
}
```

Bundle类型

- Bundle是个类型安全的容器，它的实现其实是对HashMap做了一层封装。
- HashMap中，任何名值对都可以存进去，值可以是任何的Java对象。
- 但Bundle存的名值对中，值只能是基本类型或基本类型数组。如：String、Int、byte、boolean、char等。

事件监听器的使用

- 在一个Android 应用程序中，用户与应用程序之间的交互是通过事件处理来完成的，因此，事件处理是必不可少的。
- 在进行用户界面里的事件中，进行监听的方法就是从与用户交互的特定视图对象截获这些事件。其中，视图类提供了相应的手段。
- 在各种用来组建布局的视图类里面，一些公共的回调方法对用户界面事件有用，这些方法在该对象的相关动作发生时被Android框架调用。(比如，当一个视图（如一个按钮）被触摸时，该对象上的onTouchEvent()方法会被调用等等。)

OnClickListener

- OnClickListener接口处理的是点击事件。
- 在触屏模式下，是在某个View 上按下并抬起的组合动作
- 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球事件。

- 该接口对应回调方法
  - Public void onClick（View V）；
  - 说明：参数 V 便为事件发生的事件源

OnClickListener例程-(1)

```
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)

```
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

- 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

```
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

- 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)

```
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)

```
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

- OnKeyListener 是对手机键盘进行监听的接口，通过对某个 View 注册该简体ing，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 便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)

```
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)

```
@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)

```
@Override
public boolean onKey(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

- 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)

```
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 voidonCreate（） 方法外。
```

OnTouchListener例程-(2)

```
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)

```
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;}}
```

Handler

SUN YAT-SEN UNIVERSITY

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?