操作系统

比克曼

October 23, 2015

Contents

1 android

1.1 base

- 上下文 context
  - getApplicationContext(): 生命周期是整个应用,应用摧毁,它才摧毁。
  - this: 代表当前, 在 Activity 当中就是代表当前的 Activity,换句话说就是 Activity.this 在 Activity 当中可以缩写为 this.
  - getApplication():andorid 开发中共享全局数据;

  我们在平时的开发中,有时候可能会需要一些全局数据,来让应用中得所有 Activity 和 View 都能访问到,大家在遇到这种情况时,可能首先会想到自己定义一个类,然后创建很多静态成员,不过 andorid 已经为我们提供了这种情况的解决方案:在 Android 中,有一个名为 Application 的类,我们可以在 Activity 中使用 getApplication(),方法来获得,它是代表我们的应用程序的类,使用它可以获得当前应用的主题,资源文件中的内容等,这个类更灵活的一个特性就是可以被我们继承,来添加我们自己的全局属性。

- 判断当前 Activity

```
1 ActivityManager am = (ActivityManager)
      getSystemService(ACTIVITY_SERVICE);
2 ComponentName cn = am.getRunningTasks(1).get(0).topActivity;
3 Log.d(TAG, "pkg:"+cn.getPackageName());//显示当前所在路径activity
4 Log.d(TAG, "cls:"+cn.getClassName());//路径类名+
5 Log.d(TAG, MyActivity.class.getName()); //路径类名+
6 Log.d(TAG, MyActivity.class.getSimpleName()); //类名
```

1.1.1 Bundle

android 中的 Bundle 一般用于携带数据,类似于 Map,用于存放 key-value 键值对,其提供了各种 putXx() 和 getXx() 方法,putXx() 用于往 Bundle 对象中放入数据,getXx() 用于从 Bundle 对象中获取数据。比如 Bundle 常用与组件之间进行数据传输,我们可以将 Bundle 设置好数据后,利用 Intent 的 putExtras() 方法将 Bundle 捆绑到 Intent 中,然后再传递给别的组件;

1.1.2 Intent

Intent 可以用于启动别的组件比如 Activity 和 Service 等,并在 Intent 中绑定一定的数据,传递给目标组件。

1.1.3 系统时间

- 获取年月日

```
1 import   java.text.SimpleDateFormat;
2 SimpleDateFormat formatter = new SimpleDateFormat("年月
      日yyyyMMddHH:mm:ss");
3 Date curDate = new Date(System.currentTimeMillis());//获取当前时间
4 String str = formatter.format(curDate);
```

- 获取当前的年月时分

```
1 SimpleDateFormat sDateFormat = new SimpleDateFormat("yyyy-MM-dd
      hh:mm:ss");
2 String date = sDateFormat.format(new java.util.Date());
```

- 获取当前的年月

```
SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM");
String date=sdf.format(new java.util.Date());
```

- 获取指定时区的时间

```
df =
     DateFormat.getDateTimeInstance(DateFormat.FULL,DateFormat.FULL,Locale.CHINA);
System.out.println(df.format(new Date()));
```

- 确定系统时间制式

```
ContentResolver cv = this.getContentResolver();
String strTimeFormat = android.provider.Settings.System.getString(cv,
                                                    android.provider.Settings.System.TIME_12_24)
if(strTimeFormat.equals("24")){
   Log.i("activity","24");
}
```

- 取得系统时间日期

```
Calendar c = Calendar.getInstance();
year = c.get(Calendar.YEAR) /*取得系统日期*/
month = c.grt(Calendar.MONTH)
day = c.get(Calendar.DAY_OF_MONTH)
hour = c.get(Calendar.HOUR_OF_DAY);/*取得系统时间*/
minute = c.get(Calendar.MINUTE)
```

- 利用 TIMER 获取

```
Time t=new Time(); // or Time t=new Time("GMT+8"); 加上Time 资料。Zone
t.setToNow(); // 取得系统时间。
int year = t.year;
int month = t.month;
int date = t.monthDay;
int hour = t.hour; // 0-23
int minute = t.minute;
int second = t.second;
```

1.2 activity

1.2.1 Fragment

Android 运行在各种各样的设备中,有小屏幕的手机,超大屏的平板甚至电视。针对屏幕尺寸的差距,很多情况下,都是先针对手机开发一套 App,然后拷贝一份,修改布局以适应平板神马超级大屏的。难道无法做到一个 App 可以同时适应手机和平板么,当然了,必须有啊。Fragment 的出现就是为了解决这样的问题。你可以把 Fragment 当成 Activity 的一个界面的一个组成部分,甚至 Activity 的界面可以完全有不同的 Fragment 组成,更帅气的是 Fragment 拥有自己的生命周期和接收、处理用户的事件,这样就不必在 Activity 写一堆控件的事件处理的代码了。更为重要的是,你可以动态的添加、替换和移除某个 Fragment; Fragment 必须是依存与 Activity 而存在的,因此 Activity 的生命周期会直接影响到 Fragment 的生命周期, Fragment 的生命周期见图1;

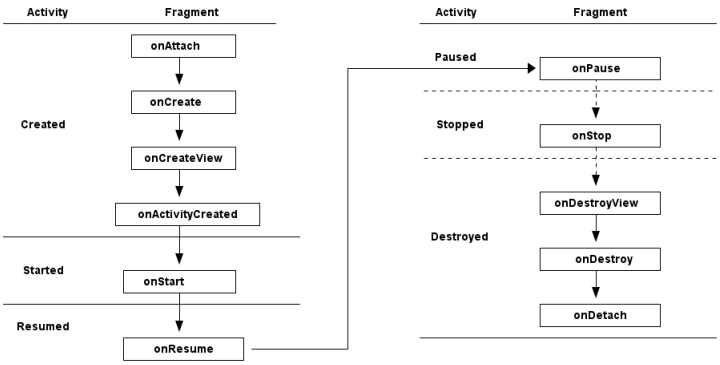

Figure 1: Fragment 生命周期

1.2.2 button

- button 透明:xml 中使用 android:background=''@android:color/transparent''

1.2.3 下拉菜单

实现某个菜单框里面有多个选项,点击后可以展示各个字符菜单项,点击后可以产生点击事件,可以将某个 int 值和字符菜单项对应;实现方法:

1. 定义两个 array 资源:

```
1 //字符菜单项
2 <string-array name="gps_type_options">
3     <item>GPS</item>
4     <item>GPS and GLONASS</item>
5     <item>GPS and BEIDOU</item>
6 </string-array>
```

```
1 //对应值int
2 <integer-array name="gps_type_values">
3     <item>0</item>
4     <item>1</item>
5     <item>2</item>
6 </integer-array>
```

2. xml 中定义器件

```
1 <Spinner
2     android:id="@+id/spinnerType"
3     android:layout_width="0dip"
4     android:layout_height="wrap_content"
5     android:layout_marginRight="8dip"
6     android:layout_weight="1" />
```

3. 在源文件中获取这 2 个资源:

```
1 String[] gpsTypeOptions =
      getResources().getStringArray(R.array.gps_type_options);
2 int[] gpsTypeValues =
      getResources().getIntArray(R.array.gps_type_values);
```

4. 绑定两者成 adapter:

```
private IntArrayAdapter mGPSTypeAdapter;
mGPSTypeAdapter = new IntArrayAdapter(this, gpsTypeOptions,
    gpsTypeValues);
```

5. 将 adapter 装配到 view 上：

```
private Spinner mSpinnerType;
mSpinnerType = (Spinner) findViewById(R.id.spinnerType);
mSpinnerType.setAdapter(mGPSTypeAdapter);
mSpinnerType.setOnItemSelectedListener(onItemSelectedListener);
```

6. 实现点击事件：

```
private OnItemSelectedListener onItemSelectedListener = new
    OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapter,View view,int
        position,long id) {
        if (adapter == mSpinnerType) {
            mGPSTypePostion = position;
        }
    }
    @Override
    public void onNothingSelected(AdapterView<?> arg0) {}
};
```

1.2.4 复选框

正方形的复选框选项

1. xml 中定义器件

```
<CheckBox
  android:id="@+id/cb"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:checked="false"
  android:text="已婚" />
```

2. 源码中获取器件

```
private CheckBox mCheckKeep;
mCheckKeep = (CheckBox) findViewById(R.id.checkKeep);
```

3. 监听事件

```
//绑定监听器
cb.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
        Toast.makeText(MyActivity.this,
         arg1?"选中了":"取消了选中"  , Toast.LENGTH_LONG).show();
    }
 });
```

4. 也可以查询获得结果

```
if(!cb.isChecked()){
}
```

1.3 service

- 在 service 中启动 activity:

```
Intent intent = new Intent(getBaseContext(), MtkPlatformTest.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)必须加这句;
startActivity(intent);
```

1.3.1 Bound Service

要做绑定服务操作,client 需要调用 bindService() , 调用后,系统将调用 server 的 onBind() 方法,这个方法将返回一个 IBinder,这个 IBinder 正是反给 client,client 使用此 IBinder 来调用 server 实现的各种服务接口,client 要取得这个 IBinder,需要实现一个接口 ServiceConnection 作为 bindService 的参数,此 ServiceConnection 中的方法 onServiceConnected 将被系统回调(在 onBind 执行完后),而 onBind 返回的 IBinder 正是作为参数传给 onServiceConnected,这样 client 就可以在 onServiceConnected 里面获得该 IBinder;

1.4 broadcast

广播机制可以事务处理异步化,可以将事务的处理放在别的地方,然后在另一个地方发送一个 Intent,系统会根据此 Intent 来找到相应的广播处理方法来处理,步骤如下。

- 设定 IntentFilter,可以在 manifest 文件中设置,也可以在源码中动态设置,实例如下;

```
private static IntentFilter makeIntentFilter() {
    final IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(ACTION_GATT_CONNECTED);
    intentFilter.addAction(ACTION_GATT_DISCONNECTED);
    intentFilter.addAction(ACTION_GATT_SERVICES_DISCOVERED);
    intentFilter.addAction(ACTION_DATA_AVAILABLE);
    return intentFilter;
}
```

- 设置广播事务处理,实例如下;

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (ACTION_GATT_CONNECTED.equals(action)) {
            //...
        } else if (ACTION_GATT_DISCONNECTED.equals(action)) {
            //...
        } else if (ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
            //...
        } else if (ACTION_DATA_AVAILABLE.equals(action)) {
            //...
        }
    }
};
```

- 注册,将 action 和事务处理相结合,实例如下;

```
registerReceiver(mReceiver, makeIntentFilter());
```

- 产生事件源,在别的地方发送消息,实例如下;

```
private void broadcastUpdate(final String action) {
    final Intent intent = new Intent(action);
    sendBroadcast(intent);
}
```

1.5 消息机制

Android 应用程序是通过消息来驱动的,系统为每一个应用程序维护一个消息队例,应用程序的主线程不断地从这个消息队例中获取消息 (Looper),然后对这些消息进行处理 (Handler),这样就实现了通过消息来驱动应用程序的执行.

- Message:消息,其中包含了消息 ID,消息处理对象以及处理的数据等,由 MessageQueue 统一列队,终由 Handler 处理。
- Handler:处理者,负责 Message 的发送及处理。使用 Handler 时,需要实现 handleMessage(Message msg) 方法来对特定的 Message 进行处理,例如更新 UI 等。
- MessageQueue:消息队列,用来存放 Handler 发送过来的消息,并按照 FIFO 规则执行。当然,存放 Message 并非实际意义的保存,而是将 Message 以链表的方式串联起来的,等待 Looper 的抽取。
- Looper:消息泵,不断地从 MessageQueue 中抽取 Message 执行。因此,一个 MessageQueue 需要一个 Looper。
- Thread:线程,负责调度整个消息循环,即消息循环的执行场所。

1.5.1 Handler

功能主要是跟 UI 线程交互用,主要有:

1. 用 handler 发送一个 message,然后在 handler 的线程中来接收、处理该消息,以避免直接在 UI 主线程中处理事务导致影响 UI 主线程的其他处理工作;
2. 你可以将 handler 对象传给其他进程,以便在其他进程中通过 handler 给你发送事件;
3. 通过 handler 的延时发送 message,可以延时处理一些事务的处理;
4. 线程处理功能:可以使用 Handler 的 post 方法,将要处理的事务放在一个 thread 里面,然后将该线程 post 到 Handler 的线程队列中 (其实这个线程和 activity 主线程是同一个线程,只是运行了线程的 run 方法),则该事务将会在 thread 里面执行,如果使用 postDelayed(thread, time) 方法,还能设置一个延时 time 后执行该事务,类似于 timer 功能;实例如下所示。

```
//使用时首先要创建一个handlerhandler
Handler handler = new Handler();
//要用来处理多线程可以使用接口,这里先定义该接口handlerrunnable
//线程中运行该接口的函数run
Runnable update_thread = new Runnable(){
    public void run(){
        //线程每次执行时输出"UpdateThread文字且自动换行...",
        //的功能和中的类似,不会覆盖前面textviewappendQtappend
        //的内容,只是中的默认是自动换行模式Qtappend
        text_view.append("\nUpdateThread...");
        //延时1后又将线程加入到线程队列中s
        handler.postDelayed(update_thread, 1000);
    }
};
```

```
15  //将线程接口立刻送到线程队列中
16  handler.post(update_thread);
17  //将接口从线程队列中移除
18  handler.removeCallbacks(update_thread);
```

5. 异步消息处理功能:同样也是使用上面线程处理功能,将某个线程 thread,post 到 handler 的线程队列中,线程队列中处理事务,并可以使用 handler 的 sendMessage(),方法向 handler 中发送 message,然后在 handler 中可以使用 handleMessage 来处理这个消息;实例如下:

```
1   //创建一个,内部完成处理消息方法handler
2   Handler update_progress_bar = new Handler(){
3       public void handleMessage(Message msg) {
4           super.handleMessage(msg);
5           //显示进度条
6           progress_bar.setProgress(msg.arg1);
7           //重新把进程加入到进程队列中
8           update_progress_bar.post(update_thread);
9       }
10  };
11  update_progress_bar.post(update_thread);//线程post
12  Runnable update_thread = new Runnable() {
13      int i = 0;
14      public void run() {
15          i += 10;
16          //首先获得一个消息结构
17          Message msg = update_progress_bar.obtainMessage();
18          //给消息结构的参数赋值arg1
19          msg.arg1 = i;
20          //延时1s
21          Thread.sleep(1000);
22          //把消息发送到消息队列中
23          update_progress_bar.sendMessage(msg);
24          if(i == 100)
25              update_progress_bar.removeCallbacks(update_thread);//移除
26      }
27  };
```

1.5.2 Looper

1.5.3 Message

1.6 menu

menu 标签中 item 标签的主要属性见表1

Table 1: android menu bar xml 属性说明

属性名	说明
android:orderInCategory	指每个 item 优先级,值越大越低,地方不够就会放到 overflow 中。
android:title	item 的标题。
android:icon	item 显示的图标。
app:showAsAction	item 显示的方式。

2 FreeRTOS

FreeRTOS 是一个迷你操作系统内核的小型嵌入式系统。作为一个轻量级的操作系统,功能包括:任务管理、时间管理、信号量、消息队列、内存管理、记录功能等,可基本满足较小系统的需要。功能和

特点:

- 混合配置选项;
- 提供一个高层次的信任代码的完整性;
- 目的是小,简单易用;
- 以开发 C,非常便携代码结构;
- 支持两项任务和共同例程;
- 强大的执行跟踪功能;
- 堆栈溢出检测;
- 没有软件任务的限制数量;
- 没有软件优先事项的限制数量;
- 没有施加的限制,优先转让,多个任务可以分配相同的优先权;
- 队列,二进制信号量,计数信号灯和递归通信和同步的任务;
- Mutexes 优先继承权;
- 免费开发工具;
- 免费嵌入式软件的源代码;
- 从一个标准的 Windows 主机交叉发展;

2.1 术语

- PV 操作:P 源自于荷兰语 parsseren,即英语的 pass;V 源自于荷兰语 verhoog,即英语的 increment。P(S)V(S) 操作是信号量的两个原子操作,S 为信号量 semaphore,相当于一个标志,可以代表一个资源,一个事件等;