回顾

1. java 面向对象
2. android 基础

基本UI,适配器ui,滑屏组件,碎片组件,导航

IO sharedprefs sqlite

app1

1. web 前端

html+css+js+jq+jqm

1. android 高级

异步处理

联网

app2 获取网络数据在移动显示处理

app3

程序组件 四大组件 后台处理消息机制

5.0以上新特性

第三方开发框架

网络框架

orm框架

第三方云平台

app4

Android 异步处理

同步:所有代码都是在一个线程执行(主线程)

异步:有一些代码是另外一个线程中执行,不在主线程中执行

android 进程/线程

进程:可以分配固定地址空间占用资源 程序都是在进程中运行 切换效率低

人民路(4车道)

线程:在一个进程中运行多个执行流程 (4车道 4个线程) 切换效率高

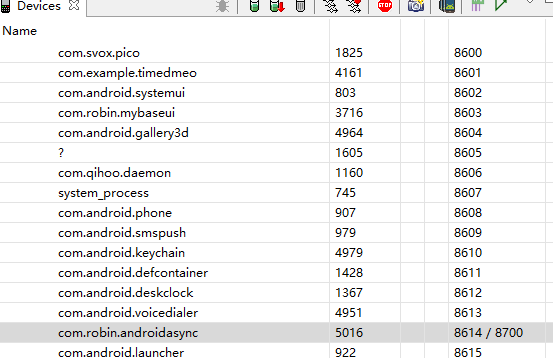
android 并发开发

1.基于java 线程来实现

2.

# android 线程

进程和线程 进程号



Pid=5016 进程号

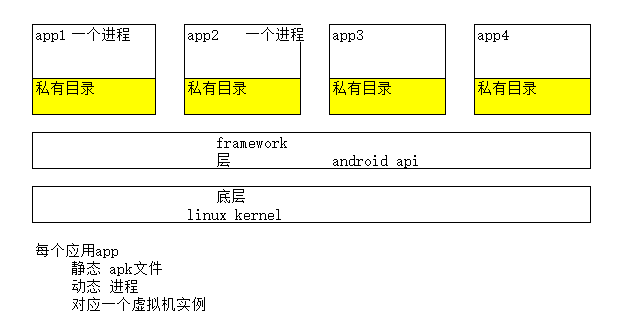
Tid=5016 主线程 tid=50

进程:android应用程序的生命周期,由线程组成.

进程之间是独立的,通常叫不同的应用.

任何一个进程内部还有一个线程没有结束,那么这个进程没有结束.

任何一个应用都有一个主线程 (永不退出,每个进程分配userid)



android 线程编程

java的并发基于线程的.

相同:基于java来定义线程的

不同:基于linux os 来实现并发,主线程不同于java主线程

android 主线程也叫UI线程,它负责处理和UI相关的事件.

Android UI不能违背两个原则:

1.当UI线程被阻塞时,超时(大约5s以上)会弹出ANR对话框.(Do not block the UI thread)

2.在UI线程以外的子线程不能访问UI对象,否则异常终止

android.view.ViewRootImpl$CalledFromWrongThreadException:

Only the original thread that created a view hierarchy can touch its views.

以上两点就是android UI线程 主线程是单线程模型的特点

## android 线程消息机制 :核心类 Handler 类

android 线程的消息机制解决线程之间通信的问题.

实现线程消息机制用到相关类有:

每个android 线程(内部基础设施 相关对象可以用来通信的)

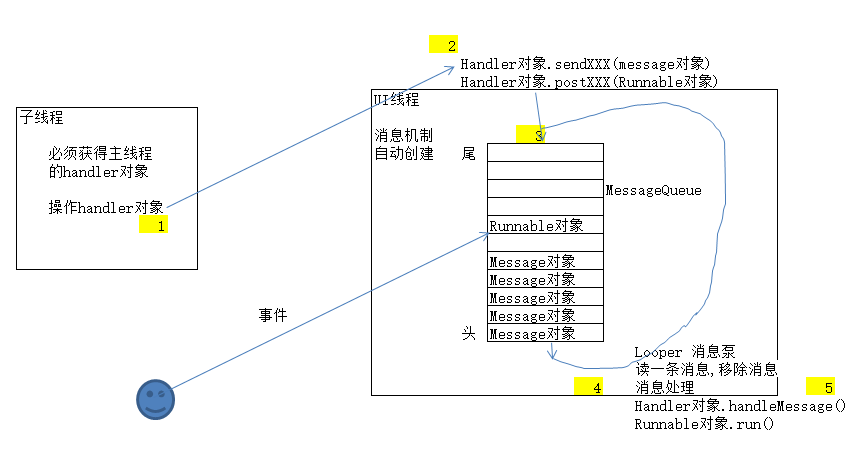
Handler

Message

MessageQueue 消息队列

Looper 消息泵

Runnable



UI线程初始化自动创建 消息队列,消息泵对象

程序员来定义Handler对象

工作原理:

1.每个线程都有一个消息队列,用来实现线程间通信,不用可忽略.

MessageQueue类

Message类

2.在线程中有一个Handler对象用来完成消息发送和消息处理.

消息发送(子线程)和消息处理(UI线程)是两个独立的部分

Handler类

3.消息的读取是由一个Looper对象用循环来执行,每读取一条消息就会回调

Handler对象的消息处理方法,并移除消息,循环读取下一条消息,直到调用

quit()退出循环为止.

子线程给UI线程发消息使用Handler步骤

1.定义 Handler子类

2.重写 handleMessage()方法,实现UI更新操作

3.创建Handler子类对象,让子线程给该对象发消息sendXXXMessage()

### 1.1.1 Handler 用途:

1.可定时或延时

启动页面

2.发消息(异步处理)

子线程异步消息机制更新Activity中的UI

//发消息对象

sendEmptyMessage(int what) 发空消息

sendMessage(Message)

sendMessageAtTime(Message, long) //在哪个时刻发

sendMessageDelayed(Message, long) //延迟多久发

//发Runnable对象

post(Runnable)

postAtTime(Runnable, long)

postDelayed(Runnable, long)

消息处理

public void handleMessage (Message msg)

获取消息

public final Message obtainMessage ()

移除消息

handler.removeMessages(456);//移除消息,但不调用消息处理方法

解释 Activity 对象原理 内部就有一个死循环

class XXXActivity extends Thread{

XXXActivity(){

//UI初始化 按钮,触屏,菜单.....

//事件处理... onXXX.....

//....

}

run(){

//死循环

while(xx){

if(hasHandler()){

//可接收各种消息....(来自于 用handler发的,也可能是用户触屏操作.....)

if(MessageQueue.getMessage()!=null){

handler.handleMessage();

}

}

}

}

onXXX(){

//....

}

}

android.text.format.Time 类似Calendar 类 效率会高些.

// 初始话一个Time对象，也可以写成:Time time = new Time("GTM+8"),即加上时区

Time time = **new** Time();

// 设置当前时间

time.setToNow();

// 获取当前时区

System.***out***.println(Time.*getCurrentTimezone*());

// 输出当前日期

System.***out***.println(time.year + "年" + time.month + "月" + time.monthDay

+ "日" + time.hour + "时" + time.minute + "分" + time.second + "秒"

+ ":现在是一年中的第" + time.yearDay + "天");

// 获取当前是一年中的第几周

System.***out***.println("现在是一年中的第" + time.getWeekNumber() + "周");

// 格式话输出当前日期、时间

System.***out***.println(time.format("%Y-%m-%d %H:%M:%S"));

System.***out***.println(time.format("%Y%m%dT%H%M%S"));// 与time.format2445相同

Studio导入工程修改的gradle设置成自己的版本:

project gradle:

buildscript {  
 repositories {  
 jcenter()  
 }  
 dependencies {  
 classpath **'com.android.tools.build:gradle:1.3.0'** *// NOTE: Do not place your application dependencies here; they belong  
 // in the individual module build.gradle files* }  
}

module app gradle文件

android {  
 compileSdkVersion 23  
 buildToolsVersion **"23.0.2"** defaultConfig {  
 applicationId **"com.robin.myasynctask"** minSdkVersion 15  
 targetSdkVersion 23  
 versionCode 1  
 versionName **"1.0"** }  
 buildTypes {  
 release {  
 minifyEnabled **false** proguardFiles getDefaultProguardFile(**'proguard-android.txt'**), **'proguard-rules.pro'** }  
 }  
}  
  
dependencies {  
 compile fileTree(dir: **'libs'**, include: [**'\*.jar'**])  
 testCompile **'junit:junit:4.12'** compile **'com.android.support:appcompat-v7:23.4.0'**}

# 2.异步处理

## 2.1 AsyncTask 异步处理核心类

### 2.1.1 异步任务处理类

异步:把代码放在另外一个线程中来执行异步操作

同步:代码都在ui线程上操作

用异步处理核心类为了简化handler操作

java.lang.Object

android.os.AsyncTask<Params, Progress, Result> 泛型类

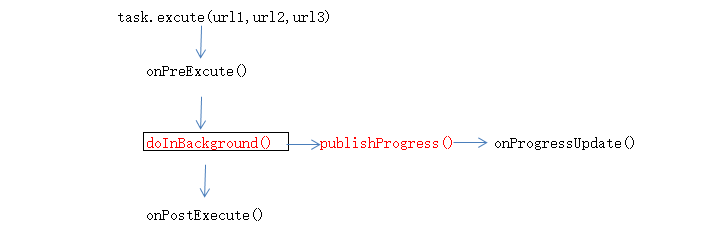
Params 输入参数的类型 URL(网址)

Progress 更新值的类型 Integer(下载进度值)

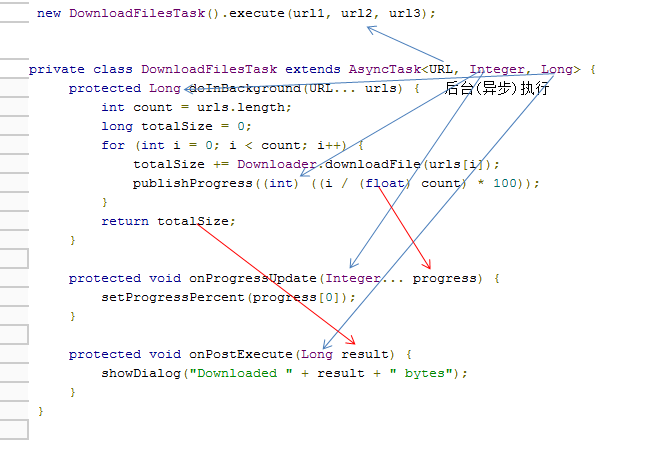
Result 操作完成值的类型 Long(下载文件多大)

作用:不用线程和handler就可实现前台(UI线程)和后台(子线程)的操作,并实现他们之间的通信.

调用顺序



参数对应关系



### 2.1.2步骤

1.定义AsynTask子类

2.重写方法 4个方法

onPreExecute() 前台处理,在调用doInBackground() 之前

doInBackground() 后台处理 (异步处理) onPreExecute()之后处理

在做后台处理时可调用publishProgress(Progress...) 告诉前台更新值的

onProgressUpdate() 前台处理,在调用publishProgress(Progress...)之后

onPostExecute() 前台处理 在doInBackground() 执行完成后

3.用该子类构造对象并调用execute()方法

private class AsyncBarTask extends AsyncTask<Void, Integer, Void>{

@Override

protected Void doInBackground(Void... params) {

// TODO Auto-generated method stub

return null;

}

@Override

protected void onProgressUpdate(Integer... values) {

// TODO Auto-generated method stub

super.onProgressUpdate(values);

}

@Override

protected void onPostExecute(Void result) {

// TODO Auto-generated method stub

super.onPostExecute(result);

}

}

Void foo(Integer … values){

Values[0]=>1

Value[1]=>2

}

Foo(1,2,3,4,5);

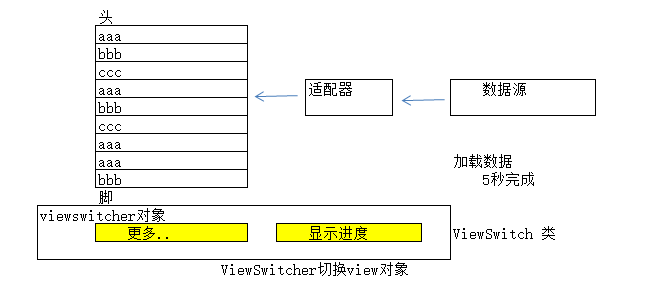
ViewSwitcher 类 切换View对象的开关

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [java.lang.Object](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\java\lang\Object.html) | | | | | |
| **↳** | [android.view.View](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\view\View.html) | | | | |
|  | **↳** | [android.view.ViewGroup](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\view\ViewGroup.html) | | | |
|  |  | **↳** | [android.widget.FrameLayout](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\widget\FrameLayout.html) | | |
|  |  |  | **↳** | [android.widget.ViewAnimator](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\widget\ViewAnimator.html) | |
|  |  |  |  | **↳** | android.widget.ViewSwitche |

AddView()

ShowNext()

showPrevious() 切换View对象



# 3.异步操作中简化用法:

## 3.1 Activity.runOnUiThread(Runnable)

## 3.2View.post(Runnable)

## 3.3View.postDelayed(Runnable, long)