目录

[1. android 线程 3](#_Toc465326389)

[1.1 android 线程消息机制 :核心类 Handler 类 5](#_Toc465326390)

[1.1.1 Handler 用途: 6](#_Toc465326391)

[2.异步处理 9](#_Toc465326392)

[2.1 AsyncTask 异步处理核心类 9](#_Toc465326393)

[2.1.1 异步任务处理类 9](#_Toc465326394)

[2.1.2步骤 10](#_Toc465326395)

[3.异步操作中简化用法: 12](#_Toc465326396)

[3.1 Activity.runOnUiThread(Runnable) 12](#_Toc465326397)

[3.2View.post(Runnable) 12](#_Toc465326398)

[3.3View.postDelayed(Runnable, long) 12](#_Toc465326399)

[4 .android 网络编程介绍 13](#_Toc465326400)

[4.1网络通信基本技术: 13](#_Toc465326401)

[4.2 api工具类 13](#_Toc465326402)

[4.3 java.net包 13](#_Toc465326403)

[4.3.1 基于TCP socket通信 14](#_Toc465326404)

[4. 2.1 Android Socket/ServerSocket 通信 16](#_Toc465326405)

[4.2.2 URL通信 19](#_Toc465326406)

[4.3 org.apache.http包 22](#_Toc465326407)

[5. Android移动端网络项目开发 22](#_Toc465326408)

[5.1联网操作 23](#_Toc465326409)

[5.1 apk文件编译流程介绍 23](#_Toc465326410)

[5.1.1获取资源文件 23](#_Toc465326411)

[5.1.2 反编译 见工具 apk改之理 23](#_Toc465326412)

[5.2 获取移动端访问的web资源 23](#_Toc465326413)

[5.3 功能模块测试 24](#_Toc465326414)

[阶段1: 24](#_Toc465326415)

[阶段2 24](#_Toc465326416)

[阶段3: 25](#_Toc465326417)

[阶段4 27](#_Toc465326418)

[5.4 app产品上线 27](#_Toc465326419)

[5.5 功能拓展 31](#_Toc465326420)

[6. WebView(网络视图) 34](#_Toc465326421)

[6.1 使用背景 34](#_Toc465326422)

[6.2 WebView 相关组件及工作原理 34](#_Toc465326423)

[6.3 WebView 类用法 35](#_Toc465326424)

[7.android 四个应用组件 37](#_Toc465326425)

[7.1 Activity 37](#_Toc465326426)

[7.1.1 活动的启动模式 38](#_Toc465326427)

[7.1.2 Intent 意图 用解决程序组件之间通信. 40](#_Toc465326428)

[7.2 BroadcastRceiver 通信机制 43](#_Toc465326429)

[7.2.1 原理 43](#_Toc465326430)

[7.2.2 广播分类 43](#_Toc465326431)

[7.2.3 广播注册 44](#_Toc465326432)

[7.2.4广播使用 44](#_Toc465326433)

[7.3 内容提供器:数据共享机制(封装数据的接口规范) 45](#_Toc465326434)

[7.3.1 原理 45](#_Toc465326435)

[7.3.2 Provider组件相关类 46](#_Toc465326436)

[7.3.3 ContentProvider 分类 46](#_Toc465326437)

[7.3.4自定义provider然后提供给别人使用的流程 46](#_Toc465326438)

[7.3.5 使用系统或别人定义的provider 50](#_Toc465326439)

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

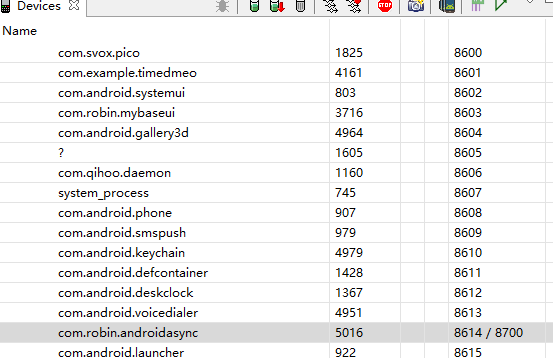
android 并发开发

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

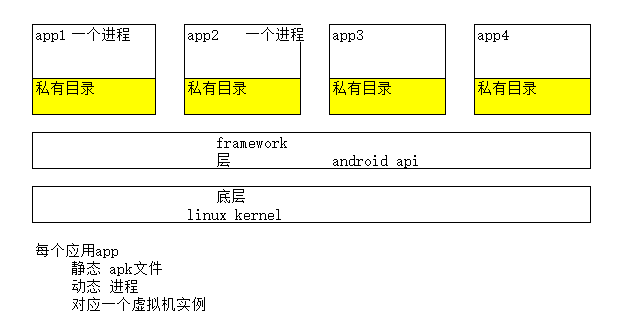
Tid=5016 主线程 tid=50

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)

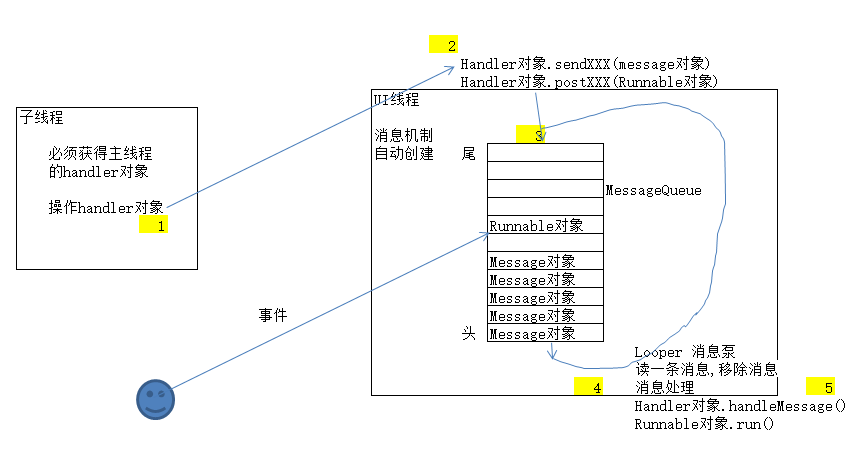
Handler

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

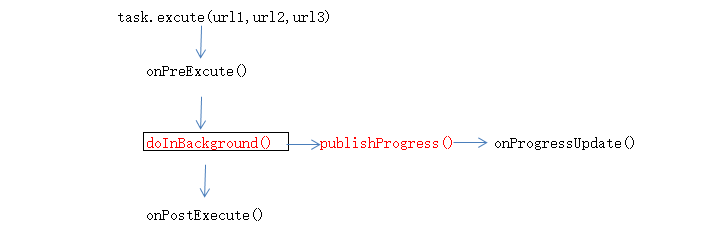
Params 输入参数的类型 URL(网址)

Progress 更新值的类型 Integer(下载进度值)

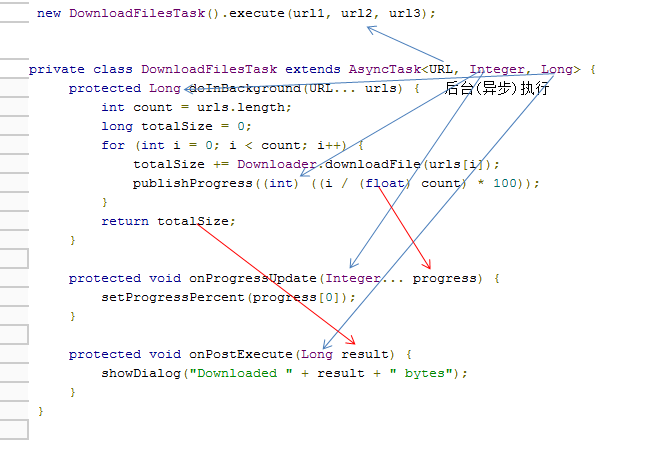
Result 操作完成值的类型 Long(下载文件多大)

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);

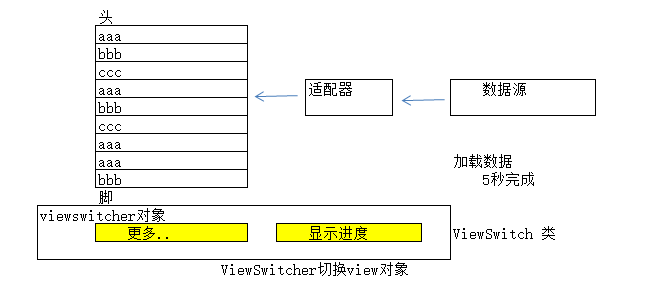
ViewSwitcher 类 切换View对象的开关

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [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



2day

回顾

1. android 进程和线程
2. android UI线程 主线程 不能两个原则
3. android 消息机制 定义Handler 子类
4. Handler 用途
5. AsyncTask 简化handler操作

# 3.异步操作中简化用法:

## 3.1 Activity.runOnUiThread(Runnable)

## 3.2View.post(Runnable)

## 3.3View.postDelayed(Runnable, long)

类似 handler.postxxx()

# 4 .android 网络编程介绍

## 4.1网络通信基本技术:

移动互联网 3G,4G,5G...(广域网)

WLAN 局域网 wifi

蓝牙短距离 NFC

socket通信

最多web通信 是web网 http协议

常用作应用客户端

## 4.2 api工具类

1.java.net包 (java 编程用的api socket通信,URL通信 重点)

2.org.apache.http包 (核心HttpClient类 了解)

3.android.net包 (辅助类 了解)

4.网络框架 volley 重点

## 4.3 java.net包

socket通信

TCP:面向连接 慢 采用一种顺序流的无差错控制的数据流 基于字节的特殊IO流(InputStream/OutputStream) 用得最广泛

类似打电话

UDP:无连接 快 有限制 数据包(报)封装数据 基于报文 响应快速的实时场景

类似发短信

socket 编程分为两种

流式socket 基于TCP实现

数据报式socket 基于UDP实现

基于TCP socket通信 两个核心类

ServerSocket 此类实现服务器套接字。

Socket 此类实现客户端套接字（也可以就叫“套接字”）。

InetAddress 此类表示互联网协议 (IP) 地址。 获得主机名,ip信息

InetSocketAddress 此类实现 IP 套接字地址（IP 地址 + 端口号）。

基于UDP socket通信

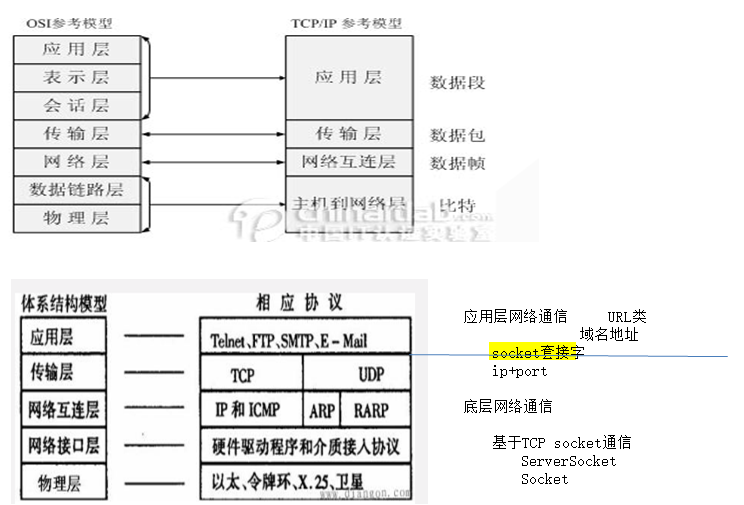
DatagramPacket 此类表示数据报包。

DatagramSocket 此类表示用来发送和接收数据报包的套接字。

与应用层通信用的

URL 类 URL 代表一个统一资源定位符，它是指向互联网“资源”的指针。

HttpURLConnection 支持 HTTP 特定功能的 URLConnection。



### 4.3.1 基于TCP socket通信

ServerSocket 此类实现服务器套接字。

端口:0-1023 系统占用

构造方法

ServerSocket(int port)

方法

Socket accept() 监听,在连接传入之前一直阻塞(线程暂停).

Socket 此类实现客户端套接字（也可以就叫“套接字”）。

构造方法

Socket(String host, int port) host:主机名/ip

方法

OutputStream getOutputStream()

InputStream getInputStream()

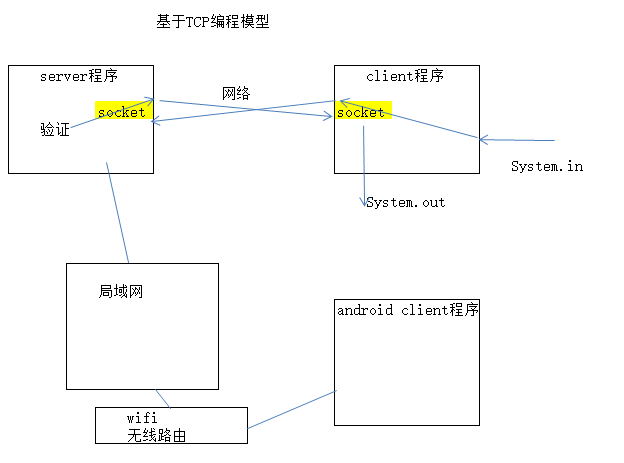
很好适配任何流.

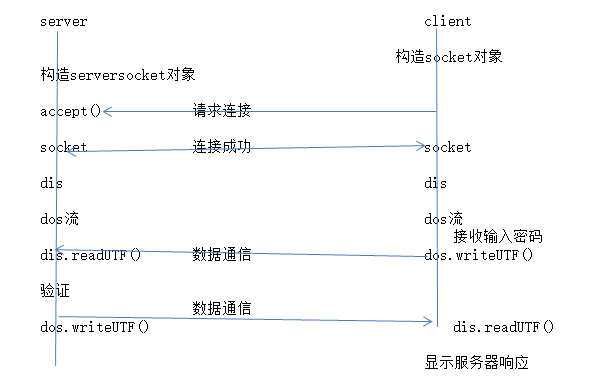
DataInputStream / DataOutputStream

readXXX() / writeXXX()

代码可能会阻塞的点

1. 创建socket连接
2. 读网络数据





TCP编程 阻塞式api

server

主线程 子线程

client

主线程

### 4. 2.1 Android Socket/ServerSocket 通信

网络api使用

使用网络必须加权限 :

<uses-permission android:name="android.permission.INTERNET" />

api版本也在变

2.x

3.x 全部采用异步访问网络否则会出现异常

4.x

5.x

java.lang.RuntimeException: Unable to start activity ComponentInfo{com.robin/com.robin.Main}:

android.os.NetworkOnMainThreadException

在主线程中使用网络会产生异常.

记住一个app如果在主线程中有网络请求操作,将会抛出该异常.

网络请求会产生阻塞主线程会ANR, 3.x以后要求网络操作必须异步请求.

解决办法:

1.StrictMode 类 强制在主线程中使用 (不推荐)

2.异步操作

Handler 框架代码

Handler handler=new Handler(){

void handleMessage(Message msg){

if(msg.what==100)

//连接网络成功.

if(msg.what=-1)

//连接失败

}

}

onCreate(){

//主线程代码

//connection();

new Thread(){

public void run(){

connection();

handler.sendEmptyMessage(100);

}

}.start();

}

connection(){

//网络请求可能会阻塞的代码

}

Android 网络编程必须做到两点:

1. 记得加权限
2. 记得将网络带有阻塞代码放到异步操作中去执行.

如何用android 客户端程序连接java tcp server?

必须知道:

1. ip=? port=911
2. 对方用的流的类型

DataInputStream DataOuputStream

readUTF() writeUTF()

3.怎么通信?

server端 用读client数据的线程来实现的 ReadThread

client 端

9patch文件

普通的图片文件经过

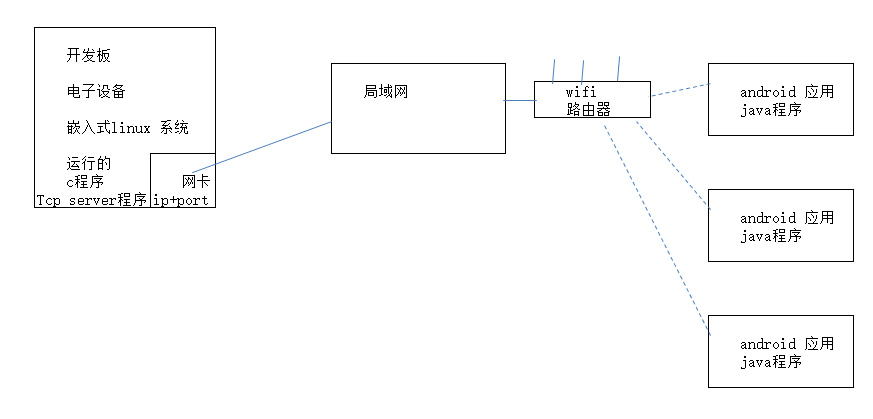
D:\zhongl\soft\android-sdk6\android-sdk\tools 下draw9patch.bat 运行对该图片进行处理

生成.9文件 可实现图片拉伸,局部保持不变形



案例

用android端应用去控制开发板设备



3day

回顾

android 网络编程

tcp socket通信(底层网络通信)

1. 加网络权限
2. 网络操作异步执行,否则产生异常

server.accept()

dis.readUTF()

new Socket()

HTTP协议用在web网络中通信,访问网络资源(绝大部分web资源)

特点:

1.http建立在TCP之上的通信协议

2.不可靠连接(短连接) http 无状态 不可靠连接

用java.net包api来实现

### 4.2.2 URL通信

URL代表访问互联网资源的地址

对于web资源表示成:

URL 可选择指定一个“端口”，它是用于建立到远程主机 TCP 连接的端口号。如果未指定该端口号，则使用协议默认的端口。例如，http 协议的默认端口为 80。还可以指定一个备用端口，如下所示：

http://www.socs.uts.edu.au:80/MosaicDocs-old/url-primer.html

<http://192.168.7.244:8080/webdata2/images/hgmm.jpeg>

URLConnection 协议无关

HttpURLConnection (支持 HTTP 特定功能的 URLConnection 和http协议相关的)

做基本socket通信

访问web资源

http协议 无状态不可靠连接 请求/响应机制

使用方法:

String actionUrl=”<http://....>”;//网址

// 定义 URL 实例

URL uri = new URL(actionUrl);//网址封装成URL对象

URLConnection con=uri.openConnection(); //协议无关连接对象

// 定义 HttpURLConnection 实例， 打开连接 访问web资源

HttpURLConnection conn = (HttpURLConnection)uri.openConnection();

//http协议连接对象 请求/响应头信息

// 设置从主机读取数据超时（ 单位： 毫秒）

conn.setReadTimeout(5 \* 1000);

// 设置允许输入

conn.setDoInput(true);

// 设置允许输出

conn.setDoOutput(true);

// 设置不允许使用缓存

conn.setUseCaches(false);

// 设置为 POST 发送方法

conn.setRequestMethod("POST");

// 设置维持长连接

conn.setRequestProperty("connection", "keep-alive");

// 设置文件字符集为 UTF-8

conn.setRequestProperty("Charsert", "UTF-8");

//发起连接请求 (可能阻塞)

conn.connect()

// 得到响应码

int res = conn.getResponseCode();

InputStream in = null;

// 上传成功则会返回响应码 200

// 200 成功 404 找不到url 500 web 后台内部错误

if (res == 200) {

// 读取数据

try {

InputStream in = new BufferedInputStream(urlConnection.getInputStream());

readStream(in);

finally {

urlConnection.disconnect();//断开连接

}

注意：如果URL传入的地址中含有中文，要对含有的中文进行URL编码，然后再拼接成地址，否则

传递数据会出现中文乱码问题，解决方式如下：

String param1 = URLEncoder.encode(中文);

请求方法

GET请求 请求参数附加url ?key1=value1&key2=value2

post请求

json解析专用网站: http://www.sojson.com/

访问web资源

1. 图片,媒体文件 二进制
2. 常见解析的数据

Json 数据

Xml 数据

Html 数据

检查更新功能的实现

1. 进入shell

nox\_adb shell

cd sdcard

ls myapp.apk

rm myapp.apk

Apk文件生成的流程

Appmodule下 gradle配置中设置版本信息:

defaultConfig {  
 applicationId **"com.robin.androidnet"** minSdkVersion 15  
 targetSdkVersion 23  
 versionCode 2  
 versionName **"2.0"**}

<http://192.168.7.10:8080/webdata2/GetJsonData?req=one> 菜单json数据

apkUrl=**"http://192.168.7.10:8080/webdata2/apk/app2.apk"**; 下载apk文件

verUrl=**"http://192.168.7.10:8080/webdata2/GetVer"**;版本信息的json数据

http://192.168.7.10:8080/webdata2/images/hgmm.jpeg

代码分析:

doUpgrade()

checkVersion();

new DownLoadTask(this, new DownLoadTask.CallBackListener() {

@Override //异步任务结束时才能调用

public void onResponse(String string) {

startLoadApk();//异步获取网络输入流

}).execute(verUrl);

startLoadApk()

new LoadStreamTask(this, new LoadStreamTask.CallBackListener() {

@Override

public void onResponse(InputStream inputStream) {

startCopyApk(inputStream);

}

}).execute(apkUrl);

startCopyApk(inputStream);

new CopyApkTask(this).execute(inputStream);

onPostExecute(){

//自动安装

}

4day

URL及相关类的使用

java.net包

## 4.3 org.apache.http包

核心接口:HttpClient

实现http客户端,具有浏览器的基本功能,但不能对页面进行解析和显示.

可实现http连接管理.

对java.net包再封装功能更为强大和简便.

Google已不再采用android studio 使用方法见

http://stackoverflow.com/questions/32153318/httpclient-wont-import-in-android-studio

okhttp

用框架来网络访问

# 5. Android移动端网络项目开发

方案1.

Web 资源:Json数据+图片 (移动端专门开发后台 生成json数据 )

方案2.

Web 资源:xml 数据+图片 (移动端专门开发后台 生成xml 数据)

方法3.

Web网站:html 数据 (不用开发后台)

## 5.1联网操作

网络端app:

获取web 资源url?

获取UI资源?

## 5.1 apk文件编译流程介绍

见docs6.0 文档 develop->tool->build system

### 5.1.1获取资源文件

### 5.1.2 反编译 见工具 apk改之理

反编译路径:

D:\1605\soft\ApkIDE3.2\ApkIDE\Work\com.robin.qsbk

1.获得 UI 资源 图片

2.布局文件

## 5.2 获取移动端访问的web资源

使用抓包工具 Fiddler

<http://www.trinea.cn/android/android-network-sniffer/>

获取web资源的url对应数据

聚合数据

获取url

分析数据

http://www.sojson.com/

完成工作:

1.将apk反编译获取资源文件

2.用抓包工具获取web资源url

Json数据

图片数据

## 5.3 功能模块测试

### 阶段1:

ListView+自定义adapter

项目:

架构(组成结构 按功能模块分)

设计分包:

com.robin.act 放activity

com.robin.adapter 放自定义adapter

com.robin.bean 放item数据值对象

com.robin.uitls 解析数据

获取UI资源

获取web 资源

每页json数据 url

每个item数据中内容图片url运算

思考问题:

1.项目用的数据在哪里? 数据是怎么表示存放?

2.项目中数据是如何展示?

3.项目中数据是如何处理的?

### 阶段2

listview+自定义adpater+asynctask+httpurlconnection+josn解析+分页加载

完成

展示

用户名

用户头像

内容信息

内容图片

上拉加载

方案1:viewswitcher

方案2:listview滚动事件监听

实现一个底部导航栏+碎片切换

底部导航栏布局

采用权重实现均匀分布

**android:layout\_width="0dp"**

**android:layout\_weight="1"**

**android:background="@drawable/tab\_image\_qiushi\_background\_selector"**

碎片操作

FragmentManager->FragmentTransaction->调用事务的方法

trans.add() 添加到布局容器并显示碎片

trans.show( ) 显示在布局容器碎片

trans.hide() 隐藏在布局容器中碎片

存在问题:

1.图片错位 保证每个item对应了一个头像图片url和一个内容图片url 对应一个item索引

2.分页加载 报异常OOM 内存溢出 OOM就是内存溢出，即Out Of Memory。也就是说内存占有量超过了VM所分配的最大。

3.缓存图片

### 阶段3:

RGB

每个像素 565格式/16位,888格式/24位 占用字节2Byte ,3Btye

每个android App其实系统只能分配一定内存空间给你运行程序

使用图片缓存技术

在应用程序的UI界面加载一张图片是一件很简单的事情，但是当你需要在界面上加载

一大堆图片的时候，情况就变得复杂起来。在很多情况下，（比如使用ListView, GridView 或者 ViewPager 这样的组件），屏幕上显示的图片可以通过滑动屏幕等事件不断地增加，最终导致OOM(java.lang.OutOfMemory异常)

为了保证内存的使用始终维持在一个合理的范围，通常会把被移除屏幕的图片进行回收处理。

此时垃圾回收器也会认为你不再持有这些图片的引用，从而对这些图片进行GC操作。

但是为了能让程序快速运行，在界面上迅速地加载图片，又必须要考虑到某些图片被回收之后，用户又将它重新滑入屏幕这种情况。这时重新去加载一遍刚刚加载过的图片无疑是性能的瓶颈，需要避免这个情况的发生。

(1)使用内存缓存技术

它可以让组件快速地重新加载和处理图片。

内存缓存技术对那些大量占用应用程序宝贵内存的图片提供了快速访问的方法。其中最核心的类是LruCache (此类在android-support-v4的包中提供) 。

这个类非常适合用来缓存图片，它的主要算法原理是把最近使用的对象用强引用存储在

LinkedHashMap 中，并且把最近最少使用的对象在缓存值达到预设定值之前从内存中移除。

为了能够选择一个合适的缓存大小给LruCache, 让一些图片常驻在内存当中

//缓存空间的大小一般设置为系统运行时内存的1/8

**int** memorySize =(**int**) Runtime.*getRuntime*().maxMemory();

**int** maxSize=memorySize/8;

int cacheSize = 4 \* 1024 \* 1024; //分配4MiB的空间cache大小

LruCache bitmapCache = new LruCache(cacheSize) {

protected int sizeOf(String key, Bitmap value) {

return value.getByteCount();

}}

synchronized (cache) {

if (cache.get(key) == null) { //从cache中取出value对象

cache.put(key, value);//将value对象保存到cahce中

}}

使用LruCache步骤:

1.计算出应用的分配大小,再来确定cache大小

2.定义存放图片到cache方法

3.定义从cache取出图片的方法

(2)使用图片三级缓存技术

缓存是否好?看到命中率LRU

内存缓存 防止内存溢出 LruCache

文件缓存

外部存储器 sd卡 /sdcard

内部存储器 flash 私有目录 /data/data/包名

内存RAM 掉电数据丢失

网络缓存

LRU是Least Recently Used 近期最少使用算法

内存置换

getView(){

loadImgForBufTask.loadImage(imageView,imageUrl);

// url->bitmap->imageView.setxxx(bitmap);

}

### 阶段4

整合

上,下导航栏,增加丑事大碎片中创建5个小碎片实现顶部导航和滑屏操作,专享小碎片展示网络的数据,其他略.

web app 上线 把开发的web 前端+后台的东西上传 服务器 给出网址

版本更新,不需要用户安装客户端程序

上线

android app 上线 做的apk上传应用市场 方便用户使用你的apk (分发效率不高)

版本更新,必须用户安装客户端程序 (热更新)

## 5.4 app产品上线

1. apk签名混淆多渠道打包

unsingend apk 是不能安装设备

经过签名

singned apk 才能安装到设备,有了签名 app防止被盗版

签名使用方法

Apk要求安装设备的条件:apk文件必须要签名

需要的三要素:

生成unsigned apk文件+你的证书(密钥)+jarsigner命令=signed apk文件

给我们自己开发的app签名，就代表着我自己的版权，以后要进行升级，也必须要使用相同的签名才行。签名就代表着自己的身份（即keystore），多个app可以使用同一个签名。

证书分为:

默认使用debug 证书

D:\zhongl\soft\android-sdk6\android-sdk\.android\debug.keystore

使用release 证书

(1)找第三方机构提供证书

(2)自定义证书

实现方法:

(1)Eclipse->export->

(2)Studio 参考 “Android Studio签名打包的两种方式”

方法1:

"Build-Generate signed apk

用debug证书签名apk

默认会在\AndroidIO\app\build\outputs\apk生成

用release(自定义)证书签名apk默认存放的路径

\AndroidIO\app生成

方法2:

配置文件+命令行工具

Gradle 工程化构建工具 生成apk文件

三个gradle配置文件

1)整个项目的gradle配置文件build.gradle 指定代码仓库jcenter

2)app文件夹下这个Module的gradle配置文件

每个module都有一个对应的gradle配置文件

apply plugin: **'com.android.application' //android 应用**android {  
 compileSdkVersion 23 //编译sdk的版本  
 buildToolsVersion **"23.0.2" //编译工具的版本** defaultConfig {  
 applicationId **"com.robin.androidio" //应用包名 必须和清单文件一致** minSdkVersion 15 //兼容的最低版本  
 targetSdkVersion 23  
 versionCode 1 //版本号  
 versionName **"1.0" //版本名 更新app时候用** }  
 buildTypes {  
 release {  
 minifyEnabled **false //对齐是否** proguardFiles getDefaultProguardFile(**'proguard-android.txt'**), **'proguard-rules.pro' //指定混淆文件** }  
 }  
}  
  
dependencies { //依赖 包有哪些  
 compile fileTree(include: [**'\*.jar'**], dir: **'libs'**)  
 testCompile **'junit:junit:4.12'** compile **'com.android.support:appcompat-v7:23.4.0'** compile **'com.jakewharton:butterknife:7.+'**}

3)settings.gradle 整个工程包含了哪些module

混淆

参见” android 通过eclipse混淆代码 打包 + proguard 总结”

把方法，字段，包和类这些java 元素的名称改成无意义的名称，这样代码结构没有变化，还可以运行，但是想弄懂代码的架构却很难。

Eclipse 混淆

Project.properties

添加:

proguard.config=proguard-project.txt

修改 proguard-project.txt

Studio 混淆

buildTypes {  
 release {  
 minifyEnabled **true** proguardFiles getDefaultProguardFile(**'proguard-android.txt'**), **'proguard-rules.pro'** signingConfig signingConfigs.release  
 }  
 }

默认module使用该**proguard-android.txt 修改此文件内容即可.**

**真正apk 发布采用第三方法加密工具**

多渠道打包 (参见Android Studio多渠道打包和代码混淆教程)

针对每个应用市场打包自己的apk,才能上传到应用市场,这时应用市场后台就可统计你的app的下载量,用户反馈,请求数等等.

1. 应用市场

作用:

由于国内Android市场众多渠道，为了统计每个渠道的下载及其它数据统计，就需要我们针对每个渠道单独打包，如果让你打几十个市场的包岂不烦死了，不过有了Gradle，这再也不是事了。

360开发者平台

http://dev.app.360.cn/

百度开放平台

http://developer.baidu.com/

应用宝开放平台

http://open.qq.com/

淘宝开放平台

http://app.taobao.com/devindex/dev\_index.htm?spm=a210u.1000832.297503.42.rNyovT

豌豆荚开发者平台

http://developer.wandoujia.com/

网易开发者平台

<http://open.t.163.com/>

身份证、身份证正反面图片、手持身份证照片、手机号码、电邮、真实姓名、软件开发者昵称、开发者版权说明书。

图标搜索网站<http://findicons.com/>

3 . 版本更新

用户检查更新

后台推送消息更新

4 . 植入广告

终端 接入移动网络

上下拉实现方案:

viewswitcher 滚动事件 xlistview

## 5.5 功能拓展

实现ListView上拉加载和下拉刷新

第三方库,控件:

Xlistview

*//设置xlistview***xListView**.setPullLoadEnable(**true**);*//启动上拉加载***xListView**.setPullRefreshEnable(**true**);*//启用下拉刷新*

**xListView**.setAdapter(**adapter**);

**xListView**.setXListViewListener(**new** XListView.IXListViewListener() {  
  
 @Override *//下拉刷新* **public void** onRefresh() {  
 **handler**.postDelayed(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **start** = ++**refreshCnt**;  
 **list**.clear();  
 getData();*//重新初始化数据源对象list  
 // mAdapter.notifyDataSetChanged();* **adapter** = **new** ArrayAdapter<String>(XListViewAct.**this**,  
 R.layout.***xlist\_simpleitem***, **list**);  
 **xListView**.setAdapter(**adapter**);  
 stop();*//停止刷新* }  
 }, 2000);  
 }  
  
 @Override *//上拉加载* **public void** onLoadMore() {  
 **handler**.postDelayed(**new** Runnable() {  
 @Override  
 **public void** run() {  
 getData();*//数据源items对象再加载数据* **adapter**.notifyDataSetChanged();  
 stop();*//停止加载* }  
 }, 2000);  
  
  
 }  
});

**private void** stop() {  
 **xListView**.stopRefresh();  
 **xListView**.stopLoadMore();  
 **xListView**.setRefreshTime(**"刚刚"**);*//显示刷新时间*}

Pulltorefresh

ViewPagerIndicator 滑屏库

cyclerviewpager 轮播

GitHub 官网(饭桶)

代码托管,版本管理(代码仓库),提供上传下载代码的网站

第三方控件,库的使用方法

1.第三方控件的定义的类和资源复制自己项目工程中 看demo学习使用的方法

2.第三方的库的使用

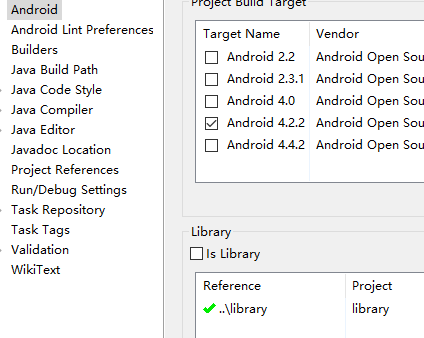
eclipse版本 跨工程调用

使用库工程library的项目工程ListSample会有:

ListSample project.properties里

android.library.reference.1=../library

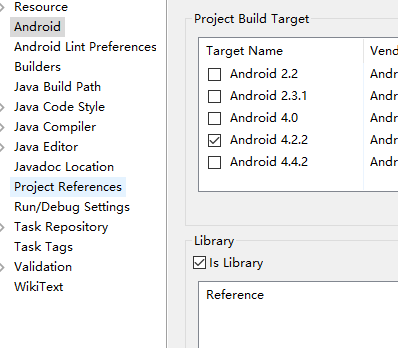
注意:建立库工程和调用的工程放在同一个目录避免出错



定义库工程的

project.properties里

android.library=true



对比eclipse 工程 类似 studio 模块module

studio版本 跨模块调用

使用库模块的项目模块

默认studio创建工程时自动生成

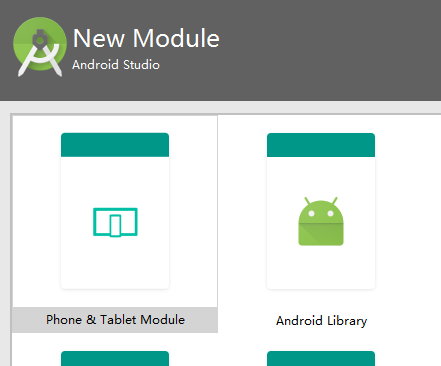
工程对应的 gradle文件 builde gradle (Project:MyPrj)

App的模块对应 gradle文件 build.gradle(Module:App)

工程对应配置gralde文件 setting.gradle(Project:settings) 说明工程中有多少module:

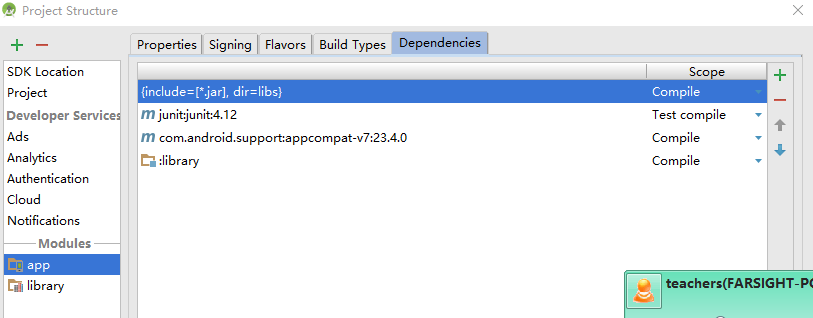
include **':app'**, **':library'**

每个xx模块都会对应gradle文件 build.gradle(Module:xx)



应用用module 库用module

应用app调用库library模块必须添加依赖:



app build.gradle里增加

dependencies {  
 compile fileTree(include: [**'\*.jar'**], dir: **'libs'**)  
 testCompile **'junit:junit:4.12'** compile **'com.android.support:appcompat-v7:23.4.0'** compile project(**':library'**)  
}

访问网络资源:

用url写代码直接获取资源 json/xml 解析

用浏览器获取资源 网页html 可自动渲染出来效果

# 6. WebView(网络视图)

## 6.1 使用背景

有时候我们可能会碰到一些比较特殊的需求，比如说要求在应用程序里展示一些网页。

但是加载和显示网页通常都是浏览器的任务，但是需求里又明确指出，不允许打开系统浏览器, 这样WebView就是一个加载显示网页，可以将其视为一个浏览器的UI控件。

## 6.2 WebView 相关组件及工作原理

内核之分

IE 稍显落后

Firefox 网景一脉相承 js之父

**Opera CSS之父**

**Chrome**

Safari webkit内核

支持html5 新标准 智能手机

核心组件 WebView

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [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.AbsoluteLayout](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\widget\AbsoluteLayout.html) | |
|  |  |  | **↳** | android.webkit.WebView |

用来实现网页浏览和用户交互接口是一个UI, 专注解析渲染工作,显示网页

辅助类

WebViewClient 提供处理各种通知,请求事件等的回调方法

WebChromeClient 提供处理js的对话框,进度条等的回调方法

WebSettings 个性化参数设置 打开js支持开关true

在Android 4.4以下(不包含4.4)系统WebView底层实现是采用

WebKit(<http://www.webkit.org/>) 内核

在Android 4.4及其以上Google 采用了chromium(<http://www.chromium.org/>)

作为系统WebView的底层内核支持。在这一变化中Android 提供的WebView相关API并没有发生大变化，在4.4上也兼容低版本的API并且引进了少部分API。

## 6.3 WebView 类用法

webview.loadUrl("http://slashdot.org/"); //加载对应url的网页

webview.loadData(summary, "text/html", null); //加载网页文档内容

webview.setWebViewClient()

webview.setWebChromeClient()

webview.getSettings()

webview.canXXX()

webview.goXXX()

使用步骤:

1.添加权限

<uses-permission android:name="android.permission.INTERNET" />

2.

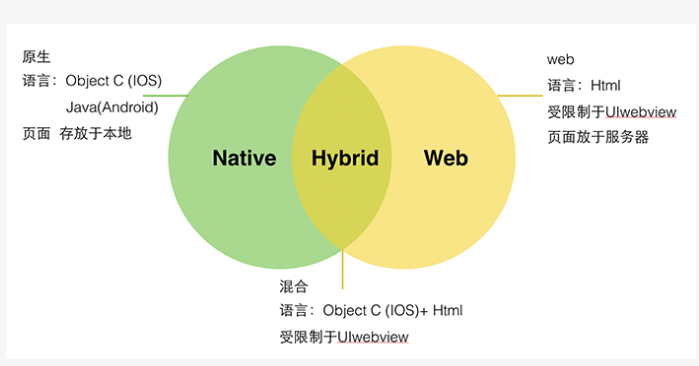
**webView**=(WebView)findViewById(R.id.***webView***);  
**webView**.getSettings().setJavaScriptEnabled(**true**);*//启动js功能***webView**.loadUrl(**"http://www.tom.com"**);  
**webView**.setWebViewClient(**new** WebViewClient() {  
 @Override *//指定url在webview中打开* **public boolean** shouldOverrideUrlLoading(WebView view, String url) {  
 view.loadUrl(url);  
 **return super**.shouldOverrideUrlLoading(view, url);  
 }  
});

@Override  
**public boolean** onKeyDown(**int** keyCode, KeyEvent event) {  
 **if** ((keyCode == KeyEvent.***KEYCODE\_BACK***) && **webView**.canGoBack()) {  
 **webView**.goBack();*//后退* **return true**;  
 }  
 **return super**.onKeyDown(keyCode, event);  
}

目前主流应用程序大体分为三类：

Web App、Hybrid App、 Native App。

http://www.uisdc.com/web-hybrid-native-app



我们做的原生android app属于 native app (纯java开发) 生成apk 安装手机上

用户体验好 事件交互响应速度 调用本设备

用jquery mobile 是(纯html+css+js 网页 ) 网页放在web服务器上 浏览器 webview

界面实现方便 分发方便版本更新方便

用hybird app webview+原生应用 最终生成apk安装手机上

andoird phonegap 核心就是 codovar 可用来打包 (网页js代码和natvie java代码交互)

Android java 如何 javascript进行交互实现方式

webview.getSetting().setJavaScriptEnable(true);

class JsObject {

**@JavascriptInterface  //高版本**

public String toString() { return "injectedObject"; }

}

webView.addJavascriptInterface(new JsObject(), "injectedObject");

//绑定 android 端用的对象 new JsObject() js端用的injectedObject

参见: android + javascript 相互通信实例分析

在做webview开发是经常会加载本机的html文件如下：

file:///android\_asset/teste.html           加载项目assets下的文件teste.html

file:///sdcard/index.html                  加载sdcard下的index.html文件

安排

1.程序组件 四大组件 activity service receiver provider 后台操作 媒体api

2.新特性 5.0

3.框架 项目

4.第三方 项目

# 7.android 四个应用组件

Activity 可见可交互 前台 界面处理

Service 不可见不可交互 后台管理(耗时任务)

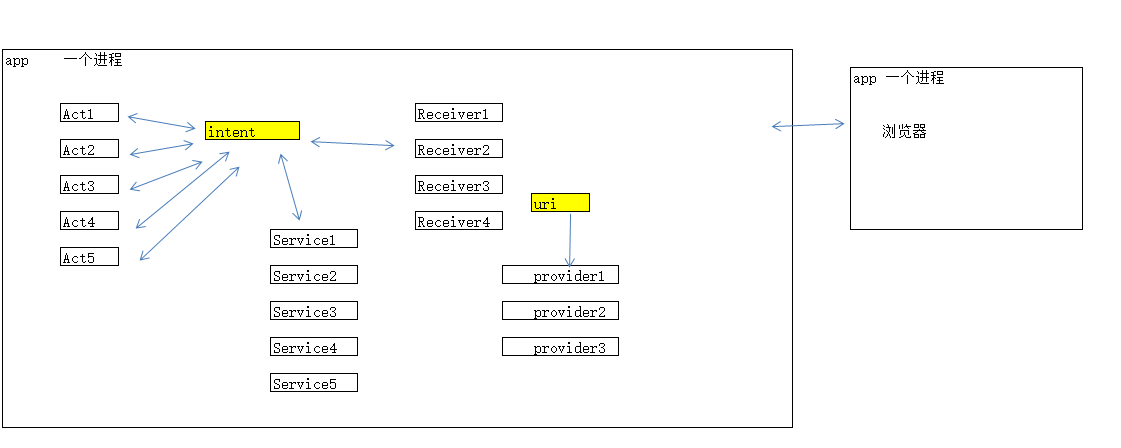
BroadcastRceiver 通信机制(活动和服务,应用和应用,应用和系统)

ContentProvider 数据共享机制 (封装数据的操作提供对外的接口)

高内聚低耦合 应用的基本骨架

所有的组件都需要靠intent对象来启动(启动某个组件,就到系统中找组件? 找不到系统报异常)

所有的组件都需要在清单文件中注册(当apk安装到android系统,系统读取清单文件中所有注册组件的信息)



所有的应用组件必须在清单文件中注册

## 7.1 Activity

1.基本用法

*//1.当apk安装android 系统时,系统会读你apk清单文件注册到系统中  
//2.在当前活动中调用startActitiy(intent) 用intent去android系统中找SecondAct活动  
// intent匹配验证 应用组件的过程 找到 启动应用组件 找不到异常退出*startActivity(**new** Intent(**this**,SecondAct.**class**));//异步方法

2.生命周期 7个回调方法提供7个回调点



1. 活动栈管理

task栈作用:

记住应用中不同活动顺序.

系统内部可以有多个task栈

默认应用X活动调用Y活动都在同一个task栈

5.活动通信

intent

intent+bundle

### 7.1.1 活动的启动模式

Activity的启动模式

类似 <a href=”” target=”self,blank,”>

启动模式的设置:

<activity android:launchMode="standard" >

1 默认启动模式 **android:launchMode="standard"**

startActivity(**new** Intent(**this**,SecondAct.**class**));//异步方法

每次启动活动会产生一个新实例,在同一个task栈

2 **android:launchMode="singleTop"**

解决栈顶多个重复相同activity的问题 “拒绝重复堆叠”

如果存在一个活动位于栈顶不产生新实例,只调用 onNewIntance()

如果活动不在栈顶会产生新实例,也在同一个task 栈中

08-17 02:43:49.190 2196-2196/? E/robin=======>: com.robin.myact.FirstAct@5281ecec

08-17 02:43:53.958 2196-2196/? E/robin=======>: com.robin.myact.SecondAct@5282d020

08-17 02:44:07.918 2196-2196/? E/robin=======>: call onNewIntent()

08-17 02:44:14.170 2196-2196/? E/robin=======>: call onNewIntent()

08-17 02:44:14.834 2196-2196/? E/robin=======>: call onNewIntent()

08-17 02:44:20.306 2196-2196/? E/robin=======>: com.robin.myact.ThirdAct@52834b0c

08-17 02:44:28.306 2196-2196/? E/robin=======>: com.robin.myact.SecondAct@5284beec

08-17 02:44:51.170 2196-2196/? E/robin=======>: call onNewIntent()

08-17 02:44:52.334 2196-2196/? E/robin=======>: call onNewIntent()

3 **android:launchMode="singleTask" 单实例**

如果存在一个活动实例位于栈顶,不产生新实例, ,只调用 onNewIntance()

如果活动不在栈顶也不会产生新实例,但位于该活动之上的所有活动会被杀死,也在同一个task 栈中

08-17 02:49:41.162 2314-2314/? E/robin=======>: com.robin.myact.FirstAct@5281fb30

08-17 02:49:52.490 2314-2314/? E/robin=======>: com.robin.myact.SecondAct@5282e500

08-17 02:50:02.274 2314-2314/? E/robin=======>: com.robin.myact.ThirdAct@528358b0

08-17 02:50:08.790 2314-2314/? E/robin=======>: com.robin.myact.FirstAct@5284baec

08-17 02:50:53.618 2314-2314/? E/robin=======>: 活动3:call onDestroy()

08-17 02:50:53.622 2314-2314/? E/robin=======>: call onNewIntent()

08-17 02:50:54.018 2314-2314/? E/robin=======>: 活动1:call onDestroy()

08-17 02:51:27.786 2314-2314/? E/robin=======>: call onNewIntent()

08-17 02:51:28.582 2314-2314/? E/robin=======>: call onNewIntent()

4 **android:launchMode="singleInstance"**

活动只在一个task栈,不能有其他实例.

08-17 03:06:35.214 2454-2454/? E/robin=======>: com.robin.myact.FirstAct@52821804

08-17 03:06:42.310 2454-2454/? E/robin=======>: com.robin.myact.SecondAct@5282e340

08-17 03:06:54.498 2454-2454/? E/robin=======>: com.robin.myact.ThirdAct@52830480

08-17 03:07:02.074 2454-2454/? E/robin=======>: com.robin.myact.FirstAct@5282a304

08-17 03:07:09.006 2454-2454/? E/robin=======>: call onNewIntent()

小结:

同一个task: 1,2,3

不同task:4

单实例:3,4 (2如果在栈顶单实例)

多实例:1

### Intent 意图 用解决程序组件之间通信.

intent用法

1.显式 指定组件类名

startActivity(**new** Intent(**this**,SecondAct.**class**));

ComponentName componentName=**new** ComponentName(**this**.getPackageName(),**"com.robin.myact.SecondAct"**);  
Intent intent=**new** Intent();  
intent.setComponent(componentName);  
startActivity(intent);

可以实现 intent进程内或进程间组件的调用,跨进程调用时组件注册要设置

**android:exported="true"**

**签名,权限,包名,组件名...**

2.隐式 用action,category,data实现验证 过滤筛选要启动的应用组件

(IntentFilter是Intent的辅助类,常用标记体现)

数据结构的组成(5要素)

action 动作字符串 "干什么"

action应用定义(程序猿做的)

action系统定义(系统提供的 [ACTION\_VIEW](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\content\Intent.html#ACTION_VIEW), [ACTION\_EDIT](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\content\Intent.html#ACTION_EDIT), [ACTION\_MAIN](file:///D:\zhongl\2016work\android\docs4.0\docs\reference\android\content\Intent.html#ACTION_MAIN))

data 用字符串来封装uri 指定"具体内容"

代表:网址,电话号码,短信...不同类型的数据封装成统一uri对象

category 类别,用来匹配条件的字符串 被请求组件增加匹配条件

type 类型,与data联合使用 mimeType

网址http:// 电话 <tel:123> 联系人***content://contacts/people/1***

component 组件名(ComponentName) 显式intent

extras 额外信息 putExtra()...

隐式用action,category,data实现验证

1.程序组件必须注册告诉系统调用我的条件是什么.

系统启动的

<**activity android:name=".FirstAct"** >  
 <**intent-filter**>//筛选条件 启动FirstAct条件是  
 <**action android:name="android.intent.action.MAIN"** />//动作app启动入口活动组件  
 <**category android:name="android.intent.category.LAUNCHER"** />//类别 程序列表  
 </**intent-filter**>  
</**activity**>

应用启动的

2.为intent设置启动程序组件需要的条件

启动程序组件成功必须以上两点.

匹配条件

1.动作验证

当有多个相同的action匹配时会弹出 intentfiler对话框

2.类别验证

intent给的类别必须和注册的组件的中某个类别匹配(添加默认类别)

3.数据验证

<data android:host="string" //主机 域名

android:mimeType="string" //互联网用数据格式 type

android:path="string" //路径

android:pathPattern="string" //前缀

android:pathPrefix="string" //匹配模式

android:port="string" //端口

android:scheme="string" /> //必须有的

hello://xxxx hello

<http://xxx> http

条件不匹配系统报异常.

使用系统的动作和类别

<intent-filter>

<action android:name="android.intent.action.MAIN" />//应用的入口指定程序组件

<action android:name="xxx" />//action可以设置多个

<category android:name="android.intent.category.LAUNCHER" />//创建到启动列表中

<category android:name="yyy" />//category可设置多个

</intent-filter>

如果有相同匹配条件系统会弹出 IntentFilter对话框

action+category+data

URI主要分三个部分：scheme, authority and path 数据格式而已.

scheme://host:port/path or pathPrefix or pathPattern

例如:

http://..... 用http协议的网址

mailto://... 用发邮件的地址

tel://... 用来打电话

file://... 文件路径

geo://..

content:// provider规定uri

scheme:没有指定,其他属性均无效.

path路径

pathPrefix 匹配前缀

hello://xxxx

http://xxx

URI主要分三个部分：scheme, authority and path 数据格式而已.

scheme://host:port/path or pathPrefix or pathPattern

例如:

http://..... 用http协议的网址

mailto://... 用发邮件的地址

tel://... 用来打电话

file://... 文件路径

geo://..

content:// provider规定uri

scheme:没有指定,其他属性均无效.

path路径

pathPrefix 匹配前缀

以上三个条件不匹配系统报异常.

实现各种系统调用参考:

http://www.cnblogs.com/sinsoul/p/3372744.html

打开另一程序

Intent i = new Intent();

ComponentName cn = new ComponentName("com.yellowbook.android2","com.yellowbook.android2.AndroidSearch");

//另一个应用包名,启动另一应用的组件名

i.setComponent(cn);

i.setAction("android.intent.action.MAIN"); //应用的入口动作

startActivityForResult(i,RESULT\_OK);

屏幕适配

当活动的配置(例如方向改变)会会导致活动销毁和重新创建

如何保存活动状态

需要定义不同资源来适配屏幕变化 layout-land layout-port

@Override *//方向发生变化才回调 保存状态***protected void** onSaveInstanceState(Bundle outState) {  
 outState.putInt(**"cnt"**,**cnt**);  
 Log.*e*(***TAG***, **"SaveIntnactState"**);  
 **super**.onSaveInstanceState(outState);  
}  
  
@Override *//方向发生变化才回调 恢复状态***protected void** onRestoreInstanceState(Bundle savedInstanceState) {  
 Log.*e*(***TAG***,**"RestoreIntnactState"**);  
  
}

## 7.2 BroadcastRceiver 通信机制

### 7.2.1 原理

广播接收器和事件处理机制类似.

事件处理程序级别(UI产生用户事件)

广播事件处理是系统级别(系统事件)

系统事件:在操作系统内部会产生一些广播.

应用事件:在app内部会产生一些广播

### 7.2.2 广播分类

1.系统广播 由系统发出

2.应用广播 由应用发出(自定义)

广播分类使用方法

1.异步广播(普通广播)

sendBroadcast(intent);//发送一个异步广播

2.有序广播

sendOrderedBroadcast(intent, null);//发有序广播

优缺点

优 1按优先级的不同，优先Receiver可对数据进行处理，并传给下一个Receiver

2通过abortBroadcast()可终止广播的传播

缺点

效率低

优先接收到Broadcast的Receiver可通过setResultExtras(Bundle)方法将处理结果存入Broadcast中，下一个Receiver 通过 Bundle bundle=getResultExtras(true)方法获取上一个 Receiver传来的数据

广播接收者需要提前设置优先级,优先级高的先接收到广播,优先级数值为

-1000~1000,在AndroidManifest.xml的

<intent-filter android:priority="" >

### 7.2.3 广播注册

1.动态注册 用代码做 适合用在应用广播(自定义的),可节省一些资源.

onCreate onDestroy

onStart onStop

onResulme onPause

onRestart

2.静态注册 用清单文件<receiver>做 适合用在系统广播,但会消耗更多资源

### 7.2.4广播使用

发广播

收广播:接收器对象

创建BroadcastReceiver步骤

1.定义BroadcastReceiver的子类,重写生命周期onReceive()方法.

2.在清单文件注册BroadcastReceiver

java.lang.Object

? android.content.BroadcastReceiver

广播生命周期

BroadcastReceiver对象在onReceiver()方法返回被销毁,生命周期短.

onReceiver 不适合做异步处理,不能阻塞

系统发

自己收

## 7.3 内容提供器:数据共享机制(封装数据的接口规范)

### 7.3.1 原理

provider实现的功能

跨应用的数据操作:CRUD操作,是一个数据库的代理.

实现跨应用间的数据共享.

类似于window服务本身不负责启动和关闭,多个应用可同时访问统一provider

使用的好处

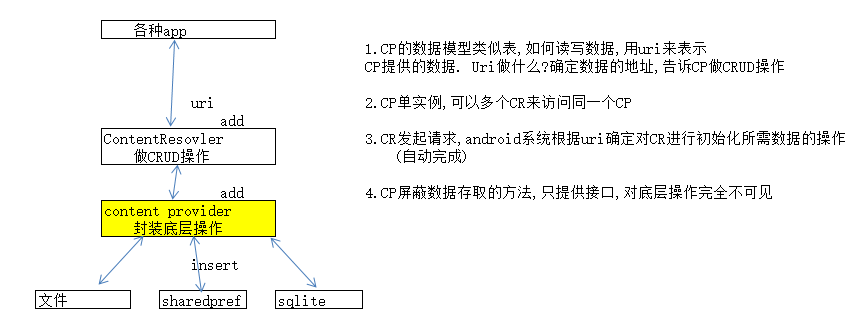
1.应用私有目录下的数据可供其他应用访问 类似java private成员 setter/getter接口

2.应用的内部的数据库非常的复杂,隐藏复杂性只提供接口

3.保证安全设置权限

如何使用provider

工作原理





想定义一个uri?

上面uir组成部分

通过uri可以获得provider的类型 getType()

定义单条记录操作uri:

content://com.example.transportationprovider/trains/122

协议 授权(区分不同cp) 表 id

调用

getType(uri)返回MIME 类型字符串

vnd.android.cursor.item/vnd.example.rail

定义多条记录操作uri

content://com.example.transportationprovider/trains

调用

getType(uri)返回MIME 类型字符串

vnd.android.cursor.dir/vnd.example.rail

小结:

定义uri相关数据需要做哪些?

单条uri 类型

多条uri 类型

### 7.3.2 Provider组件相关类

ContentProvider 类

CP对象将程序内部数据向其他程序公开

ContentResolver 类

其他程序通过CR对象对程序内部数据进行CRUD操作

### 7.3.3 ContentProvider 分类

1.使用别人的provider 比如电话本,短信,媒体 这些数据系统底层使用xx.db封装为CP接口

2.应用创建的provider 程序员自定义的,可提供给别人使用

### 7.3.4自定义provider然后提供给别人使用的流程

1.定义自己provider (定义CP的子类)

2.做一个调用自己provider测试程序(获得CR的对象,用CR来访问CP对象)

定义自己provider

1.继承ContentProvider 定义子类

2.重写6个方法(所有方法都需要uri,每个CP都提供对外公共uri)

CP将共享数据定义为uri

public class MenusProvider extends ContentProvider{

@Override

public boolean onCreate() {

// 底层数据库初始化

return false;

}

@Override

public Cursor query(Uri uri, String[] projection, String selection,

String[] selectionArgs, String sortOrder) {

// TODO Auto-generated method stub

return null;

}

@Override //要实现该方法必须完成

//1.定义MIME类型

//2.定义URI字符串

//3.用urimather建立code和uri映射

//4.用code来映射不同MIEM类型

public String getType(Uri uri) {

// 返回MIME类型,一般用静态块来完成

return null;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

// TODO Auto-generated method stub

return null;

}

@Override

public int delete(Uri uri, String selection, String[] selectionArgs) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int update(Uri uri, ContentValues values, String selection,

String[] selectionArgs) {

// TODO Auto-generated method stub

return 0;

}

}

uri代表地址,解决如何表示数据的问题.

3.声明CONTENT\_URI公共的静态常量,实现UriMathcer的绑定

当自定义provider需要告诉使用CP的人用到那些uri

UriMatcher类 将uri和MIME绑定的

ContentUris类 将uri解析/生成

(1).定义MIME

前缀

单条:

vnd.android.cursor.item

多条:

vnd.android.cursor.dir

(2).uri定义

content://<authority>/<data path>/...../id

定义authority 常量

定义data path

表名:menus

单条: "menus/#" #代表数字id 该路径下id单条记录

多条: "menus/" 对应表 该路径多条记录

"menus/\*" 代表文本 例如按名称访问

(3)解析uir的初始化  
*//1.定义code用来找对应uri的MIME类型 MULTIPLE\_MENUS=1 SINGLE\_MENUS=2  
//2.uri和code绑定以便解析*

private static final int PEOPLE = 1;

private static final int PEOPLE\_ID = 2;

private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO\_MATCH);

sURIMatcher.addURI("contacts", "people", PEOPLE);

// authority,datapath,code

URIMatcher.addURI("contacts", "people/#", PEOPLE\_ID);

public String getType(Uri url)

{

int match = sURIMatcher.match(url);//从uri中取出code

switch (match)

{

case PEOPLE:

return "vnd.android.cursor.dir/person";//MIME 多条

case PEOPLE\_ID:

return "vnd.android.cursor.item/person";//MIME 单条

}

}

生成自己使用的uri了

查询多条记录URI:Menus.***CONTENT\_URI***;

查询单条记录uri:Menus.***CONTENT\_URI***+**"/"**+id

4.实现provider底层封装到sqlite框架中.

4.1 onCreat() 初始化数据库 SQLiteOpenHelper辅助类

4.2 provider重写提供CRUD操作方法,如何使用uri

添加记录返回uri

add data:content://myfs.pub.menusprovider/menus/1

add data:content://myfs.pub.menusprovider/menus/2

add data:content://myfs.pub.menusprovider/menus/3

content://myfs.pub.menusprovider/menus/2

list={"menus","2"}

5.注册provider

<provider

android:name="com.robin.myprovider.MenusProvider"

android:authorities="myfs.pub.menusprovider" //指定授权者

android:exported="true"//跨应用共享数据 设置为true默认为false

//表示该provider能够被其他应用程序组件调用

></provider>

6.测试provider

本地测试provider

远程测试provider

表示系统中已经有同名的provider

Installation error: INSTALL\_FAILED\_CONFLICTING\_PROVIDER

观察者模式

监听CP数据改变

ContentObserver 类

当CP共享出来后,有时应用需要实时获得CP共享数据的变化需要利用

ContentObserver对象来实现

实现步骤

1.provider端的CRUD操作改变数据时调用

resolver. notifyChange()

2.使用provider端要注册CO对象

2.1 定义ContentObserver 子类重写onChange(boolean selfChange)方法

2.2 构造子类对象并注册给使用provider端的CR对象

cr.registerContentObserver(Menus.CONTENT\_URI, notifyForDescendents, observer);

Menus.CONTENT\_URI:监听CP的uri content://authority/menus

notifyForDescendents:true 该uri下所有的数据变化都监听

false 只是监听uri数据变化

### 7.3.5 使用系统或别人定义的provider

使用系统提供的provider

联系人

媒体库 (图片,音乐,视频等)

底层数据都存放在sqlite数据库中

/data/data/com.android.providers.xxxx

系统应用的数据

电话本 .contact

android.provider.ContactsContract.Contacts.CONTENT\_URI

访问电话本的数据

//显示姓名

uri=ContactsContract.Contacts.***CONTENT\_URI***;

resolver=getContentResolver();

Cursor cursor=resolver.query(uri, null, null, null, null);

Cursor对应的字段名 ContactsContract.Contacts.DISPLAY\_NAME 联系人姓名

//显示姓名和电话uri=Uri.*withAppendedPath*(ContactsContract.***AUTHORITY\_URI***,"data/phones");

Cursor cursor=resolver.query(uri, null, null, null, null);

Cursor对应的字段名 ContactsContract.Contacts.DISPLAY\_NAME 联系人姓名

ContactsContract.Contacts.Data.DATA1 联系人电话

媒体库 .media

多媒体的数据库操作类,提供一个统一接口 contentprovider

MediaStore 类 包括所有媒体相关数据的类(内部和外部存储器)

定义大量数据常量

android.provider.MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI

MediaStore.Audio

MediaStore.Audio.Media

//1.查询媒体库

Uri uri=MediaStore.Audio.Media.***EXTERNAL\_CONTENT\_URI***;

ContentResolver resolver=getContentResolver();

Cursor cursor=resolver.query(uri, **null**, **null**, **null**,MediaStore.Audio.Media.***DEFAULT\_SORT\_ORDER***);

//2.添加到媒体库

MediaScannerConnection.*scanFile*(**this**,

**new** String[] { file.getAbsolutePath() }, **null**,

**new** MediaScannerConnection.OnScanCompletedListener() {

@Override

**public** **void** onScanCompleted(String path, Uri uri) {

// **TODO** Auto-generated method stub

Log.*e*(***TAG***, "file:"+file.getAbsolutePath()+" was scanned successfully:"+uri);

}

}

);

//3. 删除媒体

Uri uri=Audio.Media.***EXTERNAL\_CONTENT\_URI***;

resolver.delete(uri,Audio.Media.***DATA***+"=?",**new** String[]{file.getAbsoluteFile().toString()});

String s=”xxxx”;//GBK UTF-8

String newstring=new String(s.getBytes(“ISO-8859-1”),”GBK”)

使用别人的provider

1.调用notelist的 NoteEdit活动

我的activity启动我的条件

data:content://com.google.provider.NotePad/notes/2

action:

<action android:name="android.intent.action.EDIT" />

<data android:mimeType="vnd.android.cursor.item/vnd.google.note" /> //MIME类型

//需要验证uri是否为该类型 调用getType(uri)==>vnd.android.cursor.item/vnd.google.note

启动该activity需要具备的条件

intent.setAction("android.intent.action.EDIT")

uri=Uri.parse("content://com.google.provider.NotePad/notes/2")

intent.setData(uri);

2.直接调用NoteList的Provider

Cursor cursor=getContentResolver().query(mUri, PROJECTION, null,null,null);