## Android 常用属性

Android：maxlength 常用在edittext中，指可以输入的最大 字符数。

Android：intype 指输入类型：有以下值

text普通文本

textCapCharacters大写字符

textCapWords单词首字母大写

textCapSentences句子首字母大写

textAutoCorret自动更正

textAutoComplete自动完成

textMultiLine多行内容

textUri，Uri

textEmailAddress电子邮件地址

textEmailSubject电子邮件主题

textShortMessage短消息

textLongMessage长消息

textPersonName个人姓名

textPostalAddress邮政地址

textPassword密码

textVIsiblePassword可见的密码

textWebEditText网页格式

textFilter过滤字符串

textPhonetic语言发音

number数字

numberSigned有符号数字

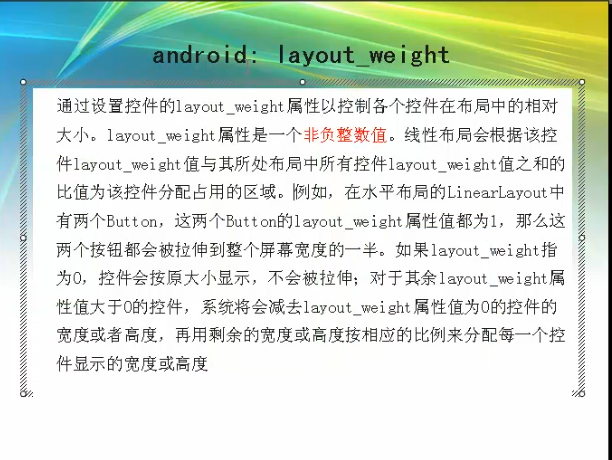
numberDecimal十进制数字

phone电话号码

android：singleline 指单行显示。

Android：hint 指示信息。

Android：layout\_weight



如果一个activity里有两个控件 ，一个weight=“2“

另一个控件的weight=“1”

系统为第一个控件分配的空间为2/1+2;

## Android程序调试：



Android：targetpackage的值必须与当前activity的包名一样。

## **创建一个LinearLayout布局，**[**LayoutParams 用法**](http://zhidao.baidu.com/question/344701106.html)。

（ambow\_android\_02\_01\_03） 02 07

其实这个LayoutParams类是用于child view（子视图） 向 parent view（父视图）传达自己的意愿的一个东西（孩子想变成什么样向其父亲说明）其实子视图父视图可以简单理解成   
一个LinearLayout 和 这个LinearLayout里边一个 TextView 的关系 TextView 就算LinearLayout的子视图 child view 。需要注意的是LayoutParams只是ViewGroup的一个内部类这里边这个也就是ViewGroup里边这个LayoutParams类是 base class 基类实际上每个不同的ViewGroup都有自己的LayoutParams子类

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

LinearLayout layout = **new** LinearLayout(**this**);

setContentView(layout);

layout.setOrientation(LinearLayout.*VERTICAL*);

**final** TextView show = **new** TextView(**this**);

Button bn = **new** Button(**this**);

bn.setText(R.string.*ok*);

bn.setLayoutParams(**new** ViewGroup.LayoutParams(

ViewGroup.LayoutParams.*WRAP\_CONTENT*,

ViewGroup.LayoutParams.*WRAP\_CONTENT*));

layout.addView(show);

layout.addView(bn);

//复杂的布局，还可能会用到一部分代码，一部分xml配置

bn.setOnClickListener(**new** OnClickListener(){

**public** **void** onClick(View arg0) {

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

show.setText("Hello,Android," + **new** java.util.Date());

}

});

}

获得当前时间和日期：java.util.Date();

**private** LayoutParams mLayoutParams = **new** LinearLayout.LayoutParams(

LinearLayout.LayoutParams.*WRAP\_CONTENT*,

LinearLayout.LayoutParams.*WRAP\_CONTENT*);

moreLayout = **new** LinearLayout(**this**);

Button btnMore = **new** Button(**this**);

btnMore.setText("load more...");

moreLayout.addView(btnMore, mLayoutParams);

addview（）;中的mlayoutparams是布局模式，即layoutparams

2,

android布局中，很多标签的属性都允许有多个值，但是需要注意的是，有的属性值之间是用|分割，有的属性值之间使用,分割…

android:gravity="right|center\_vertical"

当 android:orientation="vertical" 时， 只有水平方向的gravity设置才起作用，垂直方向的设置不起作用。即：left，right，center\_horizontal 是生效的

当 android:orientation="horizontal" 时， 只有垂直方向gravity的设置才起作用，水平方向的设置不起作用。即：top，bottom，center\_vertical 是生效的

如果一个控件没有子控件，为它设置android：gravity则设置的是它的内容的位置。

Android:gravity 指的是它的子控件或内容（没有子控件）的位置，android：layout\_gravity 指的是它在其父容器中的位置。

如果有子控件则是它的子控件的位置。

## 3，TableLayout

android:shrinkColumns 设置该列的所有单元格的宽度可以被收缩，以保证该表格能够适应父容器宽度

android:stretchColumns 设置该列的所有单元格的宽度可以被拉伸，以保证组件能完全填满表格空余空间

android:collapseColumns 设置该列的所有单元格会被隐藏

<TableLayout

android:id=*"@+id/TableLayout01"*

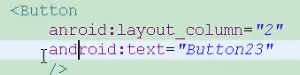
android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

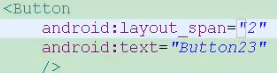
android:shrinkColumns=*"1"*

android:stretchColumns=*"2"*

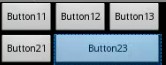
android:shrinkColumns=*"1">*



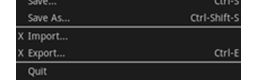
Android：layout\_column=”2” 指强制的把这个button放到第三列。



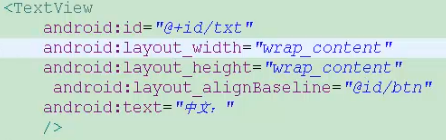
Android:layout\_span= “2” 指这个button 占两列（即合并单元格）





功能：画一条线，

可以把它写到Tablelayout布局的两个tableRow之间。



Android:layout\_alignbaseline=”@id/btn” 指textview的内容与button中线对齐 如果找不到id就在id前加+号

## 4，RelativeLayout

android:layout\_centerHorizontal 控制该组件是否位于布局容器的水平居中位置

android:layout\_centerVertical 控制该组件是否位于布局容器的垂直居中位置

android:layout\_centerInParent 控制该组件是否位于布局容器的中央位置

android:layout\_alignParentBottom 控制该组件是否与布局容器底端对齐

android:layout\_alignParentLeft 控制该组件是否与布局容器左边对齐

android:layout\_alignParentRight 控制该组件是否与布局容器右边对齐

android:layout\_alignParentTop 控制该组件是否与布局容器顶端对齐

android:layout\_toRightOf 控制该组件位于给出的ID组件的右侧

android:layout\_toLeftOf 控制该组件位于给出的ID组件的左侧

android:layout\_above 控制该组件位于给出的ID组件的上方

android:layout\_below 控制该组件位于给出的ID组件的下方

android:layout\_alignTop 控制该组件与给出的ID组件的上边界对齐

android:layout\_alignBottom 控制该组件与给出的ID组件的下边界对齐

android:layout\_alignLeft 控制该组件与给出的ID组件的左边界对齐

android:layout\_alignRight 控制该组件与给出的ID组件的右边界对齐-->

<TextView

android:id=*"@+id/tv02"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:layout\_above=*"@id/tv01"*

android:layout\_alignLeft=*"@id/tv01"*

android:background=*"@drawable/icon"* />

<!-- 定义该组件位于tv01组件的下方 -->

如果 android：layout\_alignLeft=”” 里边的值不是引用 的Id,而是某个图片像android：layout\_alignLeft=”@drewable/icon”

则表示这张图片在当前Textview的左侧。



## Textview

字体颜色

Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 这种效果



7, <!-- 设置中间省略 -->

<TextView android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:singleLine=*"true"*

android:text=*"@string/txt2"* android:ellipsize=*"middle"* />

singleLine： 设置单行显示。

Ellipsize ： 省略位置。

<TextView android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:text=*"@string/txt7"*

android:drawableLeft=*"@drawable/icon"* />

带图片的文字，

Drawableleft :设置图片位置。

<TextView android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:singleLine=*"true"*

android:text=*"@string/txt3"* android:autoLink=*"email"* />

autolink ：自动完成 、邮件链接。

Autolink的值 ：none

Web 链接到网页

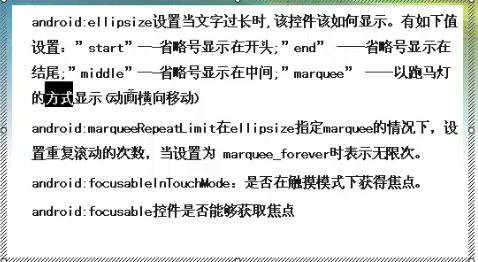
Email 链接到邮箱

Phone 链接到拨号界面

Map

All 链接到所有可用的连接。

## 跑马灯





Android:focusable=”true” 得到焦点

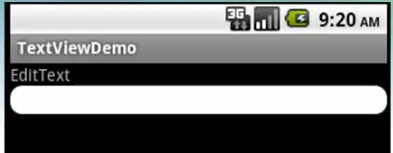
和android:focusableintouchmode=”true” 触摸时得到焦点 必须要有，一个都不能少

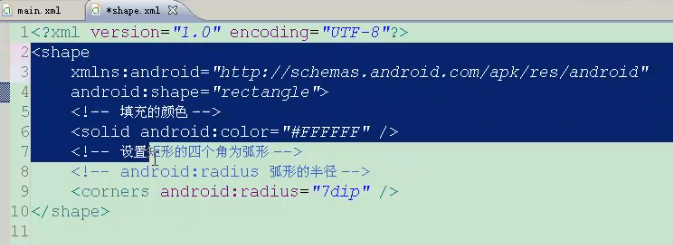
## EditText



在background 中引用shape

文本框是有弧度的像





8，phonenumber框，得到焦点时自动选择文本框内所有内容

<EditText android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:hint=*"@string/edtUserName"*

android:selectAllOnFocus=*"true"* />

<!-- android:selectAllOnFocus，设置当前组件在得到焦点的时候，自动选取该组件内的所有的文本内容 -->

<EditText android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:hint=*"@string/edtPhone"*

android:selectAllOnFocus=*"true"* android:phoneNumber=*"true"* />

phonenumber :只能写数字。。

## Button

单击事件：



35,自定义一个button的onclick方法

**public** **void** loadMore(View view) {

loadMoreButton.setText("loading...");

handler.postDelayed(**new** Runnable() {

@Override

**public** **void** run() {

**int** count = adapter.getCount();

**for** (**int** i = count; i < count + 10; i++) {

adapter.addItem(String.*valueOf*(i + 1));

}

adapter.notifyDataSetChanged();

loadMoreButton.setText("load more...");

}

}, 2000);

}

Xml文件中加一个属性onclick

<Button

android:id=*"@+id/btnMore"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:onClick=*"loadMore" loadmore是自定义的方法*

android:text=*"load more..."* />

ToogleButton

<ToggleButton

android:id=*"@+id/toggle"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:textOff=*"@string/layoutH" 按下*

android:textOn=*"@string/layoutV"* /> 抬起

监听事件

tb.setOnCheckedChangeListener(**new** OnCheckedChangeListener() {

**public** **void** onCheckedChanged(CompoundButton buttonView,

**boolean** isChecked) {

// 手工设定输出的优先级别

// Log.i("", "");//普通信息

// Log.d("", "");//debug调试

// Log.w("", "");//warn警告

// Log.e("", "");//error错误

Log.*i*(*TAG*, "onCheckedChanged");

**if** (isChecked) {

Log.*i*(*TAG*, "true");

layout.setOrientation(LinearLayout.*HORIZONTAL*);

} **else** {

Log.*i*(*TAG*, "false");

layout.setOrientation(LinearLayout.*VERTICAL*);

}

}

});

## 5，AbsoluteLayout

px像素，每个px对应屏幕上的一个点

dip或dp设备独立像素

sp比例像素

in英寸

mm毫米

pt磅，1/72英寸

<!-- 定义一个文本编辑框，使用绝对定位 -->

<EditText

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:layout\_x=*"80dip"*

android:layout\_y=*"75dip"*

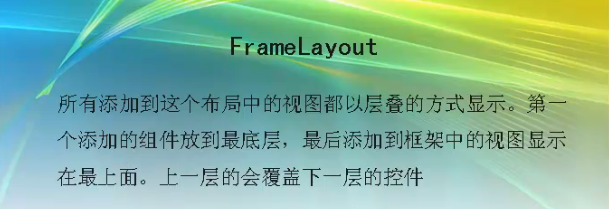
android:hint=*""*

android:password=*"true"*

android:width=*"200dip"* />

hint 提示提示信息。

Framelayout 框架布局



## 6，xml里调用布局文件

<include layout=*"@layout/other"*/> 可以使某些资源文件或UI组件可重用

## 9，imagebutton 按扭特效

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<selector xmlns:android=*"http://schemas.android.com/apk/res/android"*>

<!-- 指定按钮按钮下时的图片 -->

<item android:state\_pressed=*"true"*

android:drawable=*"@drawable/red"*

/>

<!-- 指定按钮松开时的图片 -->

<item android:state\_pressed=*"false"*

android:drawable=*"@drawable/purple"*

/>

</selector>

android:src=*"@drawable/button\_selector"*

*设置按下图片效果*

## 10，单选，复选

单选框：

<!-- 定义一组单选框 -->

<RadioGroup android:orientation=*"vertical"*

android:layout\_gravity=*"center\_horizontal"*>

<!-- 定义两个单选框 -->

<RadioButton android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:text=*"@string/male"*/>

<RadioButton android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:text=*"@string/female"* />

<RadioButton android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:text=*"@string/unknown"* />

</RadioGroup>

复选框：

<!-- 定义一个垂直的线性布局 -->

<LinearLayout android:layout\_gravity=*"center\_horizontal"*

android:orientation=*"vertical"* android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*>

<!-- 定义三个复选框 -->

<CheckBox android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:text=*"@string/red"*

android:checked=*"true"* />

<CheckBox android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:text=*"@string/blue"* />

<CheckBox android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:text=*"@string/green"* />

</LinearLayout>

## 11，Log 打印级别

// 手工设定输出的优先级别

// Log.i("", "");//普通信息

// Log.d("", "");//debug调试

// Log.w("", "");//warn警告

// Log.e("", "");//error错误

## 12，计时器 秒表

// 改变显示的信息内容,使用"%s"表示计时信息

chro.setFormat("计时信息：%s");

**public** **void** onClick(View arg0) {

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

// 设置开始计时时间

// SystemClock系统时钟类，elapsedRealtime方法是得到当前系统的真实时间

chro.setBase(SystemClock.*elapsedRealtime*());

// 启动计时器

chro.start();

Log.*i*("TestActivity", "click");

Chro.start();启动计时器，

Chro.close();停止计时器

## 15,自动完成文本框 AutoCompleteTextView

<!-- 定义一个自动完成文本框 ，指定输入一个字符后进行提示 -->

<!-- android:completionHint，设置出现在下拉菜单中的提示标题 -->

<!-- android:dropDownWidth， 设置下拉菜单的宽度-->

<!-- android:dropDownHorizontalOffset，设置下拉菜单与文本框之间的水平偏移 -->

<!-- android:completionThreshold，设置用户至少输入几个字符才会显示提示 -->

<AutoCompleteTextView android:id=*"@+id/auto"*

android:layout\_width=*"fill\_parent"* android:layout\_height=*"wrap\_content"*

android:completionHint=*"@string/title"* android:dropDownWidth=*"100dp"*

android:dropDownHorizontalOffset=*"30dp"* android:completionThreshold=*"2"* />

## 16,简单下拉列表框 Spinner

<!-- 定义了一个Spinner组件， 指定该显示该Spinner组件的数组 -->

<!-- android:entries，使用数组资源设置该下拉列表框的列表项目 -->

<!-- android:prompt，设置该列表选择框的提示 -->

<Spinner android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:entries=*"@array/books"*

android:prompt=*"@string/info"* />

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<resources>

<string-array name=*"books"*>

<item>Java SE</item>

<item>Java EE</item>

<item>Java ME</item>

<item>Linux</item>

</string-array>

</resources>

## 17，水平进度条 、可拖动的进度条 ProgressBar SeekBar

<!-- 定义一个水平进度条 -->

<ProgressBar android:id=*"@+id/bar"* android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:max=*"100"* android:progress=*"30"*

style=*"@android:style/Widget.ProgressBar.Horizontal"* />

max：最大值

progress：初始值

style：风格

seekbar用法

seekBar.setOnSeekBarChangeListener(**new** OnSeekBarChangeListener() {

// 当拖动条的滑块位置发生改变时触发该方法

**public** **void** onProgressChanged(SeekBar arg0, **int** progress,

**boolean** fromUser) {

Log.*i*(*TAG*, "onProgressChanged");

// 动态改变图片的透明度

image.setAlpha(progress);

}

//按下

**public** **void** onStartTrackingTouch(SeekBar bar) {

Log.*i*(*TAG*, "onStartTrackingTouch");

}

//抬起

**public** **void** onStopTrackingTouch(SeekBar bar) {

Log.*i*(*TAG*, "onStopTrackingTouch");

}

});

水平滑块

<SeekBar

android:id=*"@+id/seekbar"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:max=*"255"*

android:progress=*"120"*

android:thumb=*"@drawable/marker"* />

thumb:滑块图标

## 20，星级评分条

<!-- 定义一个星级评分条 -->

<RatingBar android:id=*"@+id/rating"* android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:numStars=*"5"* android:max=*"50"*

android:progress=*"10"* android:stepSize=*".3"* />

numstarts ：星星的个数

max：最大值

progress：一个星星的分值

stepsize：步长，（0-1）之间

## 21，选项卡 要继承TabActivity

TabHost tabHost = getTabHost();

// 设置使用TabHost布局

LayoutInflater.*from*(**this**).inflate(R.layout.*main*,

tabHost.getTabContentView(), **true**);

//findViewById(R.id.tab01)解析出来的组件：从布局文件中取得单独一个组件

//LinearLayout ll = findViewById(R.id.tab01);

//26行解析出来的组件：是包含在tab01中的所有的组件，靠的是17行的LayoutInFlater的inflate方法实现的。

// 添加第一个标签页

tabHost.addTab(tabHost.newTabSpec("t1").setIndicator("软件")

.setContent(R.id.*tab01*));

// 添加第二个标签页

tabHost.addTab(tabHost

.newTabSpec("t2")

// 在标签标题上放置图标

.setIndicator("网络",

getResources().getDrawable(R.drawable.*icon*))

.setContent(R.id.*tab02*));

// 添加第三个标签页

tabHost.addTab(tabHost.newTabSpec("t3").setIndicator("动漫")

.setContent(R.id.*tab03*));

}

## 22，滚动条

|  |  |
| --- | --- |
| android:scrollbars | 设置滚动显示，可以为一下一个或多个值：  none不显示滚动条  horizontal只显示水平滚动条  vertical只显示垂直滚动条 |

<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent" android:layout\_height="fill\_parent" android:scrollbars="none">

<!-- 定义HorizontalScrollView，为里面的组件添加水平滚动条 -->

<HorizontalScrollView android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content">

<LinearLayout android:orientation="vertical"

android:layout\_width="fill\_parent" android:layout\_height="fill\_parent">

<TextView android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" android:text="滚动视图滚动视图滚动视图"

android:textSize="30dp" />

</LinearLayout>

</HorizontalScrollView>

</ScrollView>

## **14，简单adapter的使用**

ambow\_android\_02\_03\_08 02 08

AutoCompleteTextView actv = (AutoCompleteTextView) findViewById(R.id.*auto*);

//适配器类

//<T>

//Context上下文

// 创建一个ArrayAdapter，封装数据

ArrayAdapter<String> adapter = **new** ArrayAdapter<String>(**this**,

android.R.layout.*simple\_dropdown\_item\_1line*, books);

// 设置Adapter

actv.setAdapter(adapter);

## 23,Listview

<!-- 直接使用数组资源给出列表项 -->

<ListView android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:entries=*"@array/books"*

android:divider=*"@drawable/red"* />

<!-- 使用ArrayAdapter提供列表项的ListView -->

<ListView android:id=*"@+id/list2"* android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:divider=*"@drawable/green"* />

ListView list2 = (ListView) findViewById(R.id.*list2*);

// 定义一个数组

String[] arr = { "大师兄", "二师兄", "三师弟" };

// 将数组包装ArrayAdapter

ArrayAdapter<String> arrayAdapter = **new** ArrayAdapter<String>(**this**,

android.R.layout.*simple\_list\_item\_1*, arr);

// 为ListView设置Adapter

list2.setAdapter(arrayAdapter);

重写onListItemClick方法

@Override

**protected** **void** onListItemClick(ListView l, View v, **int** position, **long** id) {

Toast.*makeText*(**this**, items[position], Toast.*LENGTH\_LONG*).show();

}

<ListView android:id=*"@+id/android:list"* android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"* android:background=*"#0000ff"*

android:drawSelectorOnTop=*"false"* android:choiceMode=*"multipleChoice"*/>

drewselectorontop=是否覆盖文字

choicemode 单选和复选框，只是外观。

String[] arr = { "刘德华", "张学友", "黎明", "郭富城" };

List<HashMap<String, Object>> data = **new** ArrayList<HashMap<String, Object>>();

**for** (**int** i = 0; i < arr.length; i++) {

HashMap<String, Object> item = **new** HashMap<String, Object>();

item.put("num1", i);

item.put("value1", arr[i]);

data.add(item);

}

/\*

\* 1、simple\_list\_item\_1，每一个列表项都是一个普通的TextView

\* 2、simple\_list\_item\_2，每一个列表项都是一个普通的TextView，字体略大，TextView显示标题，TextView显示内容，因此

\* 需要两组数据，并且指定每组数据和两个TextView的对应关系

\* 3、simple\_list\_item\_checked，每一个列表项都是一个已勾选的列表项

\* 4、simple\_list\_item\_multiple\_choice，每一个列表项都是带多选项

\* 5、simple\_list\_item\_single\_choice，每一个列表项都是带单选项

\*/

// ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,

// android.R.layout.simple\_list\_item\_2, arr);

SimpleAdapter adapter = **new** SimpleAdapter(**this**, data,android.R.layout.*simple\_list\_item\_2*,

**new** String[]{"num1","value1"}, **new** **int**[]{android.R.id.*text1*,android.R.id.*text2*});

// 设置该窗口显示列表

setListAdapter(adapter);}

## 自定义adapter

自定义arrayadapter风格

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="horizontal" >

<!-- 在每一个元素中（每一行list中），有一个image icon，然后是我们的数据信息 -->

<ImageView

android:id="@+id/image"

android:layout\_width="44px"

android:layout\_height="wrap\_content"

android:src="@drawable/ic\_launcher" />

<TextView

android:id="@+id/label"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="44sp" />

</LinearLayout>

setListAdapter(**new** ArrayAdapter<String>(**this**, R.layout.*item*,

R.id.*label*, items));

自定义适配器 extends arrayadapter

**private** **class** MyAdapter **extends** ArrayAdapter<String> {

MyAdapter() {

**super**(TestActivity.**this**, R.layout.*item*, items);

}

//当利用adapter加载数据的时候，getView方法，用来给List生成每一个列表项目，在这个示例中，由于

//每一个列表项目由多个组件组成，并且，组件的值有特殊的要求，因此，必须重写getView方法。

@Override

**public** View getView(**int** position, View convertView, ViewGroup parent) {

//Inflater,LayoutInflater,MenuInflater

LayoutInflater infalter = getLayoutInflater();

View row = infalter.inflate(R.layout.*item*, parent, **false**);

TextView label = (TextView) row.findViewById(R.id.*label*);

ImageView icon = (ImageView) row.findViewById(R.id.*image*);

label.setText(items[position]);//第一次为jack，以此类推。

**if** (position > 3) {

icon.setImageResource(R.drawable.*a*);

} **else** {

icon.setImageResource(R.drawable.*b*);

}

**return** row;

}

}

自定义适配器 extends baseadapter

// 自定义的Adapter类

**private** **class** ListViewAdapter **extends** BaseAdapter {

// 为组件提供数据的。

**private** List<String> items;

**private** LayoutInflater inflater;

**public** ListViewAdapter(Context context, List<String> items) {

**this**.items = items;

inflater = getLayoutInflater();

// inflater = LayoutInflater.from(TestActivity.this);

}

@Override

**public** **int** getCount() {

**return** items.size();

}

@Override

**public** Object getItem(**int** position) {

**return** items.get(position);

}

@Override

**public** **long** getItemId(**int** position) {

**return** position;

}

@Override

**public** View getView(**int** position, View view, ViewGroup parent) {

**if** (view == **null**) {

view = inflater.inflate(R.layout.*item*, **null**);

}

TextView text = (TextView) view.findViewById(R.id.*label*);

text.setText(items.get(position));

**return** view;

}

View loadMoreView = getLayoutInflater().inflate(R.layout.*more*, **null**);

Button loadMoreButton=(Button)loadMoreView.findViewById(R.id.*btnMore*);

//getListView()的作用是返回当前的ListView对象（id：list）

// getListView().addFooterView(moreLayout);

getListView().addHeaderView(loadMoreView);

setListAdapter(**new** MyAdapter());

foorerview 加到当前listview的最后，

headerview加到当前listview 的前面。

## 24,设置 时间、日期

<EditText android:id=*"@+id/show"* android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:editable=*"false"*

android:cursorVisible=*"true"* />

android:editable 是否可编辑

android：cursorvisible 光标是否可见

DatePicker datePicker = (DatePicker) findViewById(R.id.*datePicker*);

TimePicker timePicker = (TimePicker) findViewById(R.id.*timePicker*);

// 获取当前的年、月、日、小时、分钟

Calendar c = Calendar.*getInstance*();

//Student s = new Student();原始社会，自己创建，找一块石头，打磨斧头

//Student s = Factory.getInstance(int type);工业社会，不需要自己创建，到工厂中，需要什么，买什么,1石斧，2钢斧

//Spring--->IOC共产主义社会

//Student s ;在家坐在沙发看电视，

year = c.get(Calendar.*YEAR*);

month = c.get(Calendar.*MONTH*);

day = c.get(Calendar.*DAY\_OF\_MONTH*);

hour = c.get(Calendar.*HOUR*);

minute = c.get(Calendar.*MINUTE*);

// 初始化DatePicker组件，初始化时指定监听器

datePicker.init(year, month, day, **new** OnDateChangedListener() {

**public** **void** onDateChanged(DatePicker arg0, **int** year, **int** month,

**int** day) {

TestActivity.**this**.year = year;

TestActivity.**this**.month = month;

TestActivity.**this**.day = day;

// 显示当前日期、时间

showDate(year, month, day, hour, minute);

}

});

// 为TimePicker指定监听器

timePicker.setOnTimeChangedListener(**new** OnTimeChangedListener() {

**public** **void** onTimeChanged(TimePicker arg0, **int** hour, **int** minute) {

TestActivity.**this**.hour = hour;

TestActivity.**this**.minute = minute;

// 显示当前日期、时间

showDate(year, month, day, hour, minute);

}

});

}

// 定义在EditText中显示当前日期、时间的方法

**private** **void** showDate(**int** year, **int** month, **int** day, **int** hour, **int** minute) {

EditText show = (EditText) findViewById(R.id.*show*);

show.setText("您的购买日期为：" + year + "年" + month + "月" + day + "日 " + hour

+ "时" + minute + "分");

//允许自定义组件

}

## 26，list列表中添加图片和其它信息

// 创建一个List集合，List集合的元素是Map

List<Map<String, Object>> listItems = **new** ArrayList<Map<String, Object>>();

**for** (**int** i = 0; i < names.length; i++) {

Map<String, Object> listItem = **new** HashMap<String, Object>();

listItem.put("header", imageIds[i]);

listItem.put("personName", names[i]);

listItem.put("personInfo", values[i]);

listItems.add(listItem);

}

// 创建一个SimpleAdapter

SimpleAdapter simpleAdapter = **new** SimpleAdapter(**this**, listItems,

R.layout.*main*, **new** String[] { "personName", "header" ,"personInfo"},

**new** **int**[] { R.id.*name*, R.id.*header*,R.id.*info* });

ListView list = (ListView) findViewById(R.id.*mylist*);

// 为ListView设置Adapter

list.setAdapter(simpleAdapter);

}

<ListView android:id=*"@+id/mylist"* android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* />

<!-- 定义一个ImageView，用于作为列表项的一部分。 -->

<ImageView android:id=*"@+id/header"* android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* />

<!-- 定义一个TextView，用于作为列表项的一部分。 -->

<TextView android:id=*"@+id/name"* android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:textSize=*"16dp"*/>

<TextView android:id=*"@+id/info"* android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"* android:textSize=*"16dp"*></TextView>

## **27，gridview**

ambow\_android\_02\_03\_20 02 09

<GridView android:id=*"@+id/grid01"* android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:numColumns=*"4"* />

<!-- 定义一个ImageSwitcher组件 -->

<ImageSwitcher android:id=*"@+id/switcher"*

android:layout\_width=*"320dp"* android:layout\_height=*"320dp"*/>

numcolumns: 列数。

//在android中大部分组件的数据，都由adapter提供服务

// 创建一个List对象，List对象的元素是Map

List<Map<String, Object>> listItems = **new** ArrayList<Map<String, Object>>();

**for** (**int** i = 0; i < imageIds.length; i++) {

Map<String, Object> listItem = **new** HashMap<String, Object>();

listItem.put("image", imageIds[i]);

listItem.put("info", infos[i]);

listItems.add(listItem);

}

// 获取显示图片的ImageSwitcher

**final** ImageSwitcher switcher = (ImageSwitcher) findViewById(R.id.*switcher*);

//动画效果可以自定义，也可以引用系统自带的动画效果。

//Animation是android提供的动画效果类

//xxxUtils,AnimationUtils,StringUtils,EncodingUtils

//loadAnimation为组件附加（加载）动画效果

// 设置图片更换的动画效果

switcher.setInAnimation(AnimationUtils.*loadAnimation*(**this**,

android.R.anim.*fade\_in*));

switcher.setOutAnimation(AnimationUtils.*loadAnimation*(**this**,

android.R.anim.*fade\_out*));

//为switcher设定工厂类，作用是创建一个动态的ImageView，显示用户所选择的图片

switcher.setFactory(**new** ViewFactory() {

@Override

**public** View makeView() {

ImageView imageView = **new** ImageView(TestActivity.**this**);

imageView.setBackgroundColor(0xff0000);

imageView.setLayoutParams(**new** ImageSwitcher.LayoutParams(

LayoutParams.*WRAP\_CONTENT*, LayoutParams.*WRAP\_CONTENT*));

**return** imageView;

}

});

// 创建一个SimpleAdapter

SimpleAdapter simpleAdapter = **new** SimpleAdapter(**this**,

listItems

// 使用/layout/cell.xml文件作为界面布局

, R.layout.*cell*, **new** String[] { "image" ,"info"},

**new** **int**[] { R.id.*image1* ,R.id.*info* });

GridView grid = (GridView) findViewById(R.id.*grid01*);

// 为GridView设置Adapter

grid.setAdapter(simpleAdapter);

// 添加列表项被选中的监听器

grid.setOnItemSelectedListener(**new** OnItemSelectedListener() {

@Override

**public** **void** onItemSelected(AdapterView<?> parent, View view,

**int** position, **long** id) {

Log.*i*(*TAG*, "" + position);

// 显示当前被选中的图片

switcher.setImageResource(imageIds[position]);

}

@Override

**public** **void** onNothingSelected(AdapterView<?> parent) {

}

});

// 添加列表项被单击的监听器

grid.setOnItemClickListener(**new** OnItemClickListener() {

@Override

**public** **void** onItemClick(AdapterView<?> parent, View view,

**int** position, **long** id) {

// 显示被单击的图片的图片

switcher.setImageResource(imageIds[position]);

}

});

## 28，AlertDialog对话框

Button bn = (Button) findViewById(R.id.*bn01*);

// 定义一个AlertDialog.Builder对象

**final** Builder builder = **new** AlertDialog.Builder(**this**);

// 为按钮绑定事件监听器

bn.setOnClickListener(**new** View.OnClickListener() {

@Override

**public** **void** onClick(View source) {

// 设置对话框的图标

builder.setIcon(R.drawable.*icon*);

// 设置对话框的标题

builder.setTitle("自定义普通对话框");

// 设置对话框显示的内容

builder.setMessage("一个简单的提示对话框");

// 为对话框设置一个“确定”按钮

builder.setPositiveButton("确定"

// 为列表项的单击事件设置监听器

, **new** OnClickListener() {

@Override

**public** **void** onClick(DialogInterface dialog,

**int** which) {

EditText show = (EditText) findViewById(R.id.*show*);

// 设置EditText内容

show.setText("用户单击了“确定”按钮！");

}

});

// 为对话框设置一个“取消”按钮

builder.setNegativeButton("取消", **new** OnClickListener() {

@Override

**public** **void** onClick(DialogInterface dialog, **int** which) {

EditText show = (EditText) findViewById(R.id.*show*);

// 设置EditText内容

show.setText("用户单击了“取消”按钮！");

}

});

builder.setNeutralButton("测试", **null**);

// 创建、并显示对话框

builder.create();

builder.show();

//也可以直接用 builder.create().show

}

});

创建一个列表对话框

// 为对话框设置多个列表

b.setItems(**new** String[] { "红色", "绿色", "蓝色" }

// 为按钮设置监听器

, **new** OnClickListener() {

// 该方法的which参数代表用户单击了那个列表项

@Override

**public** **void** onClick(DialogInterface dialog,

**int** which) {

TextView show = (TextView) findViewById(R.id.*show*);

// which代表哪个列表项被单击了

**switch** (which) {

**case** 0:

show.setBackgroundColor(Color.*RED*);

**break**;

**case** 1:

show.setBackgroundColor(Color.*GREEN*);

**break**;

**case** 2:

show.setBackgroundColor(Color.*BLUE*);

**break**;

}

}

});

// 创建、并显示对话框

b.create().show();

}

## 通过重写oncreadDialog方法创建对话框

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

Button bn = (Button) findViewById(R.id.*bn*);

// 为按钮绑定事件监听器

bn.setOnClickListener(**new** View.OnClickListener() {

@Override

**public** **void** onClick(View source) {

// 显示对话框

showDialog(FLAG);

//flag是一常量

}

});

}

// 重写onCreateDialog方法创建对话框

@Override

**public** Dialog onCreateDialog(**int** id, Bundle state) {

// 判断需要生成哪种类型的对话框

**switch** (id) {

**case** FLAG:

Builder b = **new** AlertDialog.Builder(**this**);

// 设置对话框的图标

b.setIcon(R.drawable.*icon*);

// 设置对话框的标题

b.setTitle("单选列表对话框");

// 为对话框设置多个列表

b.setSingleChoiceItems(**new** String[] { "红色", "绿色", "蓝色" }

// 默认选中第二项

, 1

// 为列表项的单击事件设置监听器

, **new** OnClickListener() {

@Override

**public** **void** onClick(DialogInterface dialog, **int** which) {

TextView show = (TextView) findViewById(R.id.*show*);

// which代表哪个列表项被单击了

**switch** (which) {

// 修改文本框的背景色

**case** 0:

show.setBackgroundColor(Color.*RED*);

**break**;

**case** 1:

show.setBackgroundColor(Color.*GREEN*);

**break**;

**case** 2:

show.setBackgroundColor(Color.*BLUE*);

**break**;

}

}

});

// 添加一个“确定”按钮，用于关闭该对话框

b.setPositiveButton("确定", **null**);

b.create().show();

// 创建对话框

**return** b.create();

}

**return** **null**;

}

单选列表对话框 要创建一个确定按扭。

## 29，Toast提示信息

Button simple = (Button) findViewById(R.id.*simple*);

// 为按钮的单击事件绑定事件监听器

simple.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View source) {

// 创建一个Toast提示信息

Toast toast = Toast.*makeText*(TestActivity.**this**, "简单的提示信息", Toast.*LENGTH\_SHORT*);

toast.show();

}

});

Button bn = (Button) findViewById(R.id.*bn*);

// 为按钮的单击事件绑定事件监听器

bn.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View source) {

// 创建一个Toast提示信息

Toast toast = Toast.*makeText*(TestActivity.**this**, "带图片的的提示信息"

// 设置该Toast提示信息的持续时间

, Toast.*LENGTH\_LONG*);

// 获取Toast提示里原有的View

View toastView = toast.getView();

// 创建一个ImageView

ImageView image = **new** ImageView(TestActivity.**this**);

image.setImageResource(R.drawable.*tools*);

//ViewGroup

// 创建一个LinearLayout容器

LinearLayout ll = **new** LinearLayout(TestActivity.**this**);

// 向LinearLayout中添加图片、原有的View

ll.addView(image);

ll.addView(toastView);

toast.setView(ll);

toast.show();

}

});

## 30，创建menu菜单

@Override

**public** **boolean** onCreateOptionsMenu(Menu menu) {

// 创建菜单的时候，被调用

Log.*i*(*TAG*, "onCreateOptionsMenu");

// groupId，将某些菜单项目放到一个组中，便于管理。

// itemId，为菜单项目指定一个唯一的id。

// order,指定菜单项目的排序。

// 显示在菜单中的文本内容。

// 如果想对创建的菜单项目进行其他编码的话，直接在add方法的后面.的方式调用即可。

menu.add(0, *ITEM1*, 1, "File");

menu.add(0, *ITEM2*, 0, "Edit");

menu.add(0, *ITEM3*, 0, "Test");

menu.add(0, 4, 0, "Test1");

menu.add(0, 5, 0, "Test2");

menu.add(0, 6, 0, "Test3");

menu.add(0, 7, 0, "Test4");

//返回给系统，告诉系统当前的方法是否已经对需要的地方进行了处理，如果已经确认处理完毕，return true，如果需要系统（父）接着做一些后续处理的话，return false；

**return** **true**;

}

@Override

**public** **boolean** onOptionsItemSelected(MenuItem item) {

// 菜单项目被选择的时候，被调用

Log.*i*(*TAG*, "onOptionsItemSelected");

**switch** (item.getItemId()) {

**case** *ITEM1*:

setTitle("File");

**break**;

**case** *ITEM2*:

setTitle("Edit");

**break**;

**case** 3:

setTitle("test1");

**break**;

}

**return** **true**;

}

## 31，创建上下文菜单（长按）

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

textView = (TextView) findViewById(R.id.*textView*);

//为某个组件注册上下文菜单

registerForContextMenu(textView);

}

@Override

**public** **void** onCreateContextMenu(ContextMenu menu, View v,

ContextMenuInfo menuInfo) {

//创建上下文菜单，在某个组件中，长按的时候触发

menu.add(0, *ITEM1*, 0, "Red");

menu.add(0, *ITEM2*, 0, "Green");

menu.add(0, *ITEM3*, 0, "White");

}

@Override

**public** **boolean** onContextItemSelected(MenuItem item) {

//在某个菜单项目被选择的时候触发

**switch**(item.getItemId()){

**case** *ITEM1*:

textView.setBackgroundColor(Color.*RED*);

**break**;

**case** *ITEM2*:

textView.setBackgroundColor(Color.*GREEN*);

**break**;

**case** *ITEM3*:

textView.setBackgroundColor(Color.*WHITE*);

**break**;

}

**return** **true**;}

## 32,submenu （子菜单项）

**public** **boolean** onCreateOptionsMenu(Menu menu) {

//SubMenu代表了子菜单项目

SubMenu file = menu.addSubMenu("File");

SubMenu edit = menu.addSubMenu("Edit");

file.add(0, *ITEM1*, 0, "New");

file.add(0, *ITEM2*, 0, "Open File");

edit.add(0,3,0,"cut");

edit.add(0,4,0,"pause");

**return** **true**;

}

@Override

**public** **boolean** onOptionsItemSelected(MenuItem item) {

**switch**(item.getItemId()){

**case** *ITEM1*:

setTitle("New");

**break**;

**case** *ITEM2*:

setTitle("Open File");

**break**;

**case** 3:

setTitle("cut");

**break**;

}

**return** **true**;

}

## 33，用menuinflater解析xml文件封装成menu对象

<!-- -menu标签代表了Menu类，item标签代表了MenuItem -->

<menu xmlns:android=*"http://schemas.android.com/apk/res/android"* >

<!-- menu.add(0,ITEM,0,"File").setIcon(R.drawable.ic\_launcher); -->

<item

android:icon=*"@drawable/ic\_launcher"*

android:title=*"File"*>

<!-- SubMenu file = menu.addSubMenu(File);file.add(0,ITEM,0,"New") -->

<menu>

<group

android:id=*"@+id/noncheckable\_group"*

android:checkableBehavior=*"none"* >

<!-- android:alphabeticShortcut属性，快捷键的设置 -->

<item

android:id=*"@+id/newFile"*

android:alphabeticShortcut=*"n"*

android:title=*"New"*/>

<item

android:id=*"@+id/openFile"*

android:alphabeticShortcut=*"o"*

android:title=*"Open"*/>

<item

android:id=*"@+id/saveFile"*

android:alphabeticShortcut=*"s"*

android:title=*"Save"*/>

</group>

</menu>

</item>

<item

android:icon=*"@drawable/ic\_launcher"*

android:title=*"Edit"*>

<menu>

<group

android:id=*"@+id/edit\_group"*

android:checkableBehavior=*"single"* >

<item

android:id=*"@+id/cut"*

android:title=*"Cut"*/>

<item

android:id=*"@+id/copy"*

android:title=*"Copy"*/>

<item

android:id=*"@+id/past"*

android:title=*"Past"*/>

</group>

</menu>

</item>

<item

android:icon=*"@drawable/ic\_launcher"*

android:title=*"Help"*>

<menu>

<group android:id=*"@+id/help\_group"* >

<item

android:id=*"@+id/about"*

android:title=*"About"*/>

<item

android:id=*"@+id/exit"*

android:title=*"Exit"*/>

</group>

</menu>

</item>

</menu>

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

mi = **new** MenuInflater(**this**);

}

@Override

**public** **boolean** onCreateOptionsMenu(Menu menu) {

mi.inflate(R.menu.*menu*, menu);//进行解析，将menu.xml中配置的菜单项目，封装成menu对象

**return** **true**;

}

@Override

**public** **boolean** onOptionsItemSelected(MenuItem item) {

**switch** (item.getItemId()) {

**case** R.id.*cut*:

Log.*i*("TestActivity",R.id.*cut*+"");

**break**;

**case** R.id.*about*:

Log.*i*("TestActivity", "about");

**break**;

**case** R.id.*exit*:

Log.*i*("TestActivity", "exit");

**break**;

}

**return** **true**;

}

## 34，intent的使用

/\*

\* Intent通过下面的属性来描述以上的某个意图：

\* 1、Action：指要完成一个抽象动作,当前的意图或目的是什么。

\* 2、Category：为Action动作属性增加额外的附加类别信息。

\* 3、Data：用于向Action动作属性提供操作的数据。Data属性接收的参数为Uri对象。 http://www.sina.com.cn,ftp://168.192.1.100,

\* 格式如下：

\* xxxx://xxxxxxxxxx

\* 其中，冒号前面解释了数据的类型，冒号后面是数据部分。

\* 例如：

\* "content:com.android.contacts/contacts/1"

\* 4、Type：明确指定Data属性所指定数据的类型，通常来说，当Intent不指定Data属性时Type属性才会起作用。

\* 否则Android系统将会根据Data属性值来分析数据类型，因此无须指定Type属性。

\* 5、Extras：通常用在多个Activity之间进行数据交换。

\* 6、Component：目标组件，封装要跳转的组件对象。

\*

\* 工作原理：当程序向Android发送一个Inent请求时，Android会根据Intent的内容在注册的IntentFilter中选择适当的组件来响应。（匹配）

\*

\* 过程：

\* 1、调用者创建Intent对象，并设置相关属性。对于隐式Intent，由于没有明确目标组件名称，所以必须包含足够的属性信息，他们是：Action，Data，Category。

\* 2、向Android提交Intent请求。

\* 3、Android对Intent进行解析，选择相应的组件进行相应。由Android系统帮助应用程序寻找与Intent请求意图最匹配的组件。

\*/

两个activity相互跳转：

@Override

**public** **void** onClick(View v) {

Intent intent = **new** Intent(AndroidTest.**this**,SecondActivity.**class**);

startActivity(intent);

}

跳转时发送信息

@Override

**public** **void** onClick(View v) {

String name = e1.getText().toString();

String address = "bj";

Bundle data = **new** Bundle();

data.putString("name", name);

data.putString("address", address);

Intent intent = **new** Intent(AndroidTest.**this**,ResultActivity.**class**);

intent.putExtras(data);

startActivity(intent);

}

接收跳转时带来的信息

Intent intent = getIntent();

Bundle b = intent.getExtras();

String name = b.getString("name");

Map<String,String> users = **new** HashMap<String,String>();

users.put("jack", "13100000000");

users.put("nina", "13111111111");

users.put("sandy", "13122222222");

String tel = users.get(name);

t1.setText(name + "的电话号码为：" + tel);

用startActivityForResult(intent, 1);跳转

Intent intent = **new** Intent(AndroidTest.**this**,

ThirdActivity.**class**);

startActivityForResult(intent, 1);

Intent intent = getIntent();

SecondActivity.**this**.setResult(2, intent);

// SecondActivity.this.setResult(0, intent);

SecondActivity.**this**.finish();

@Override

**protected** **void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {

Log.*i*("onActivityResult", String.*valueOf*(requestCode));

Log.*i*("onActivityResult", String.*valueOf*(resultCode));

**if**(requestCode == 1 && resultCode == 2){

// if(requestCode == 0){

Log.*i*("test", "button0");

}**else** **if**(requestCode == 1 && resultCode == 3){

// }else if(requestCode == 1){

Log.*i*("test", "button1");

}

}

startActivityForResult 的第二个参数是requestcode

setResult 的第一个参数是resultcode

跳转信息可以封装

ComponentName cn = **new** ComponentName(AndroidTest.**this**,

"com.ambow.android.SecondActivity");

Intent intent = **new** Intent();

intent.setComponent(cn);

startActivity(intent);

或 intent.setClass(AndroidTest.this, SecondActivity.class);

接收

Intent intent = getIntent();

ComponentName cn = intent.getComponent();

String packageName = cn.getPackageName();

String className = cn.getClassName();

String str = cn.getShortClassName();

t = (TextView) findViewById(R.id.*textView*);

t.setText("packageName:" + packageName + "&className:" + className+str);

隐式意图跳转

//android中对于隐式意图来说，系统会默认为intent添加一个category属性

intent.setAction("com.ambow.android.MY\_ACTION2");

intent.addCategory("com.ambow.android.MY\_CATEGORY");

必需要在manifest 中声明 category必须要有。

<application android:icon=*"@drawable/icon"* android:label=*"@string/app\_name"*>

<activity android:name=*".AndroidTest"*

android:label=*"@string/app\_name"*>

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

<!--

当某个Activity发送Intent的时候，android系统得到intent，对intent的属性，进行解析，得到action属性的值，和配置文件中的所有的intent-filter里面的信息，进行匹配

当匹配成功，会把匹配了的Activity作为目的地。显示出来。当匹配失败，报出没找到合适的Acitivity。

-->

<activity android:name=*".SecondActivity"*>

<intent-filter>

<action android:name=*"com.ambow.android.MY\_ACTION1"*/>

<action android:name=*"com.ambow.android.MY\_ACTION2"*/>

<category android:name=*"android.intent.category.DEFAULT"*/>

<category android:name=*"com.ambow.android.MY\_CATEGORY"*/>

</intent-filter>

</activity>

</application>

接收

Intent intent = getIntent();

String action = intent.getAction();

Set<String> categorys = intent.getCategories();

t.setText("Activity2 " + action);

**if** (categorys != **null**) {

**for** (String s : categorys) {

t.setText(t.getText() + ":" + s);

}

}

## Acction常用的值

/\*

\* Action常用值

\* ACTION\_MAIN：Android Application的入口，每个Android应用必须且只能包含一个此类型的Action声明。

\* ACTION\_VIEW：系统根据不同的Data类型，通过已注册的对应Application显示数据。

\* ACTION\_EDIT：系统根据不同的Data类型，通过已注册的对应Application编辑示数据。

\* ACTION\_DIAL：打开系统默认的拨号程序，如果Data中设置了电话号码，则自动在拨号程序中输入此号码。

\* ACTION\_CALL：直接呼叫Data中所带的号码。

\* ACTION\_SENDTO：系统根据不同的Data类型，通过已注册的对应Application进行数据发送操作。

\* 略...

\*

\* Category常用值

\* CATEGORY\_DEFAULT：Android系统中为隐式意图添加的默认的执行方式，按照普通Activity的执行方式执行。

\* CATEGORY\_HOME：设置该组件为Home Activity。

\* CATEGORY\_LAUNCHER：设置该组件为在当前应用程序启动器中优先级最高的Activity，通常为入口ACTION\_MAIN配合使用。

\* 略...

\*/

## Data常用值

/\*

\* Data常用值

\* tel://：号码数据格式，后跟电话号码。

\* smsto://：短息数据格式，后跟短信接收号码。

\* content://：内容数据格式，后跟需要读取的内容。

\* geo://latitude, longitude：经纬度数据格式，在地图上显示经纬度所指定的位置。

\* 略...

\*/

data = "content://com.android.contacts/contacts/1";查看通讯录中id为1的电话信息。

Data = "content://com.android.contacts/contacts/"查看通讯录中所有信息。

data = "http://www.sina.com.cn"; 访问网址。

data = "tel:13800138000"; 打电话。用ACTION\_CALL 直接拨号，用ACTION\_DIAL打开拨号界面。

data = "geo:381111,1161111";访问地图

data = "smsto:13800138000";发短信。

使用过程

Intent intent = new Intent();

String data = "";

Uri uri = Uri.*parse*(data);

intent.setAction(Intent.*ACTION\_DIAL*);

intent.setData(uri);

startActivity(intent);

回到主页

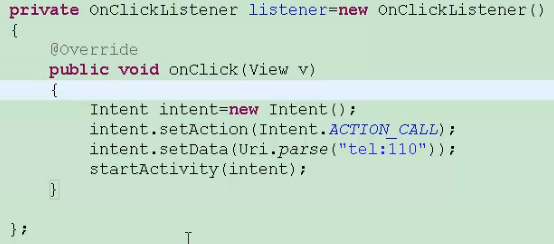
intent.setAction(Intent.*ACTION\_MAIN*);

intent.addCategory(Intent.*CATEGORY\_HOME*);

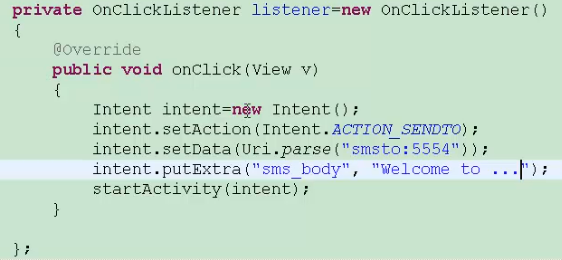
startActivity(intent);

intent 一些功能 要在manifest中添加相应的权限

打电话



发短信



## 36,Activity生命周期。

Android0506 0215

<application

android:icon=*"@drawable/icon"*

android:label=*"@string/app\_name"* >

<activity

android:name=*".AndroidTest"*

android:label=*"@string/app\_name"* >

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

<activity

android:name=*".AndroidTest2"* android:theme=*"@android:style/Theme.Dialog"*>

</activity>

</application>

android:theme=*"@android:style/Theme.Dialog"*

第二个activity会以一个窗口的方式显示。不会把第一个activity 完全覆盖，第一个activity可视，但不可操作。

首次启动一个activity时，会依次调用

Oncreate方法（首次启动首先调用）

Onstart 方法 （//当activity被用户可视，但是不能进行响应时调用）

Onresume 方法（//当activity被用户可视，并且可以进行响应时调用。）

当另外一个activity中断当前activity时

当前activity首先调用

Onpause 方法（//当一个activity被另一个activity中断的时候，调用

//保存一些重要的，可能需要较长时间保存的数据）

如果另外 一个avtivity覆盖当前activity时（即当前activity不可视）

当前activity 调用

onstop 方法（//当activity被另一个activity完全遮盖住，并且，另一个activity已经可视，已经可以和用户交互的时候。）

如果当前activity没有被另外 一个activity完全覆盖，（即可视）则当前activity不会调用onstop 方法。

当前activity被唤醒时会首先调用

Onrestart 方法

然后会调用onstart onresume 方法。

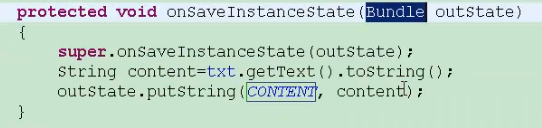
当一个activity被关掉时会依次调用

Onpause 方法

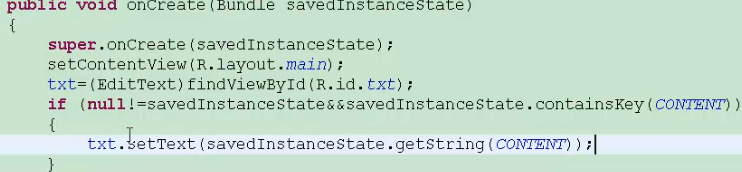
Onstop 方法

Ondestory 方法（//当系统内存不足以运行android系统的时候，会优先将停止的activity释放掉。）

Onsaveinstancestate 方法有可能在onpause前调用，也可能在onpause之后调用，可以用来保存一些临时的数据。



下次执行oncreate时取出保存的数据。



Onsaveinstancestate 中的outstate 和 oncreate 中的savedinstancestate 是同一个bundle 对象。

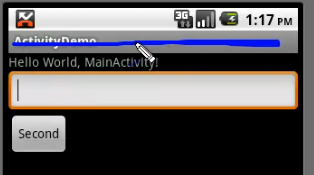
*Savedinstancestate.containskey(content) 指的是savedinstancestate中包含key 为content的值 。*

## android：theme其它方式

*每个theme前面都要加@android:style/*



*Notitlebar不显示标题*



*Notitlebar.fullscreen 应用程序占满整个程序。不显示*

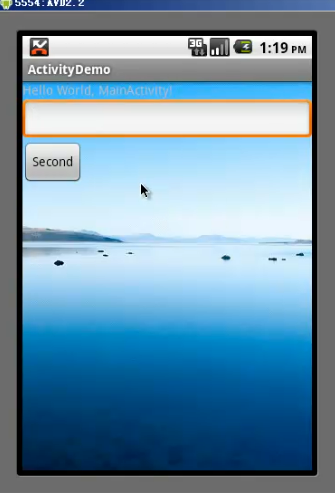


*Light 背景是白色*

*Black 背景是黑色*



*Wallpaper 背景和当前的壁纸相同。*



*Translucent 背景是透明的*

## 37，用sharedpreferences 存储数据

通过onstop方法保存数据。

//利用getSharedPreferences方法，取得SharedPreferences对象

//方法的参数：1、文件的名称2、文件的模式

//MODE\_PRIVATE：代表文件是私有的，并且当我们生成Preferences File的时候，将内容全部替换

//MODE\_WORLD\_READABLE：允许其他应用程序对他进行读取。

//MODE\_WORLD\_WRITEABLE：允许其他应用程序对他进行写入。

//MODE\_APPEND：添加数据的时候，数据以追加的方法，放到文件内容的最后。

//得到当前的应用程序所在的文件夹下的临时文件

pre = getSharedPreferences(*TEMP\_SMS*, *MODE\_WORLD\_READABLE*);

//Temp\_sms 是要保存文件的名字。

//getString方法取得数据

//方法的参数：1、要取得的数据的key2、默认值（当没有找到符合条件的数据的话，会返回该默认值）

String content = pre.getString("sms\_content", **null**);//默认值，如果是String，null，如果是数字，0

txt.setText(content);

}

@Override

**protected** **void** onStop() {

**super**.onStop();

//Editor是用来编辑PreferencesFile数据的对象。

SharedPreferences.Editor editor = pre.edit();

//putString的作用是用来写入数据的。key-value

editor.putString("sms\_content", txt.getText().toString());

editor.putString("test", "test");

//commit提交，将写入的数据，真正的提交到物理文件中。提交数据，rollback

editor.commit();

}

根据包名，取得其应用程序的上下文对象，访问其数据

其存储的数据必须要有访问权限。

//上下文对象 Button b = new Button(this);getSharedPreferences();

//根据"com.ambow.android"包名，取得其他应用程序的上下文

//在android系统中，每一个应用程序都运行在各自独立的dalvik进程中。

Context con = createPackageContext("com.ambow.android", Context.*CONTEXT\_IGNORE\_SECURITY*);

SharedPreferences sp = con.getSharedPreferences(*TEMP\_SMS*, *MODE\_WORLD\_READABLE*);

String sms = sp.getString("sms\_content", "");

title.setText(sms);

用输入输出流读写文件

写

**private** **void** write(String content){

**try**{

//openFileOutput参数1、写入的文件的名称2、写入文件的模式

FileOutputStream fos = openFileOutput(*“temp.text”*, *MODE\_APPEND*);

//write，写的操作

fos.write(content.getBytes());

fos.flush(); 提交

fos.close();

}**catch**(Exception e){

e.printStackTrace();

}

}

默认写到包名下一个名为file的文件夹下

读

**private** String read(){

**try**{

FileInputStream fis = openFileInput(*FILE\_NAME*);

**byte**[] buffer = **new** **byte**[fis.available()];

fis.read(buffer);

fis.close();

**return** **new** String(buffer);

}**catch**(Exception e){

e.printStackTrace();

}

**return** **null**;

}

### 设定字符串的格式

//设置字符串的字符集

**public** String encode(**byte**[] array) {

String s = **new** String("hello world"); //ISO-8859-1

**try** {

s = **new** String(s.getBytes("ISO-8859-1"),"UTF-8");//解码，按照原有的字符集格式，编码，可执行现有的字符集格式

} **catch** (UnsupportedEncodingException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*out*.println(s);

//xxxUtils

**return** EncodingUtils.*getString*(array, *TEXT\_ENCODING*);

}

自定义一个类封装 读写 转码的操作 android 090102 0223

File 和 fileoutputstream 配合使用

//判断当前是否有扩展的存储设备，并且对存储设备有相关的权限。

**if** (Environment.*getExternalStorageState*().equals(

Environment.*MEDIA\_MOUNTED*)) {

File sdcard = Environment.*getExternalStorageDirectory*();//mnt/sdcard

File f = **new** File(sdcard.getCanonicalPath() + "/temp.abc");//mnt/sdcard/temp.abc

// File f = new File("mnt/sdcard/temp.abc");

FileOutputStream fout = **new** FileOutputStream(f);

fout.write("hello world".getBytes());

fout.flush();

fout.close();

## 38，sqlite 数据库 增、删、改、查，及其实现

Android 090302 0224

// 数据库类，里面提供了增删改查、事务处理、数据库连接关系的方法。

// DriverManager驱动程序管理

// Connection连接对象

// Statement执行sql对象

// PreparedStatement:select \* from where id = ?

// CalllableStatement调用存储过程

// ResultSet保存数据对象结果集

打开或创建一个数据库

// 打开或者创建数据库，并且返回该数据库对象

// 参数1、数据库名称

// 参数2、数据库的类型

// 参数3、定义CursorFactory，如果我们需要自定义Cursor的时候，需要用到该参数，通常情况下，都设置成null

SQLiteDatabase db = context.openOrCreateDatabase(*DATABASE\_NAME*, Context.*MODE\_PRIVATE*,

**null**);

建表：

try {

/\*\*

\* 一般数据采用的固定的静态数据类型，而SQLite采用的是动态数据类型，会根据存入值自动判断。SQLite具有以下五种数据类型：

\* 1.NULL：空值。

\* 2.INTEGER：带符号的整型，具体取决有存入数字的范围大小。

\* 3.REAL：浮点数字，存储为8-byte IEEE浮点数。

\* 4.TEXT：字符串文本。

\* 5.BLOB：二进制对象。

\*/

//execute，执行sql语句

db.execSQL("CREATE TABLE t\_user ("

+ "\_ID INTEGER PRIMARY KEY autoincrement," + "age INTEGER,"

+ "NAME TEXT" + ");");

Log.*v*(*TAG*, "Create Table t\_user ok");

} **catch** (Exception e) {

Log.*v*(*TAG*, "Create Table t\_user err,table exists.");

}

增：

**public** **void** add(String name, String age) {

// String sql = "insert into user(age,name) values("+age+",'"+name+"')";

// database.execSQL(sql);

contentValues = **new** ContentValues();

contentValues.put("age", age);

contentValues.put("name", name);

database.insert("user", "\_id", contentValues);

}

删：

**public** **void** delete(**int** id){

String sql = "delete from user where \_id="+id;

database.execSQL(sql);

}

改：

**public** **void** update(**int** id ,String name,String age) {

contentValues = **new** ContentValues();

contentValues.put("age" , age);

contentValues.put("name", name);

database.update("user", contentValues, "\_id = ?", **new** String[]{(id + "").trim()});

}

查：

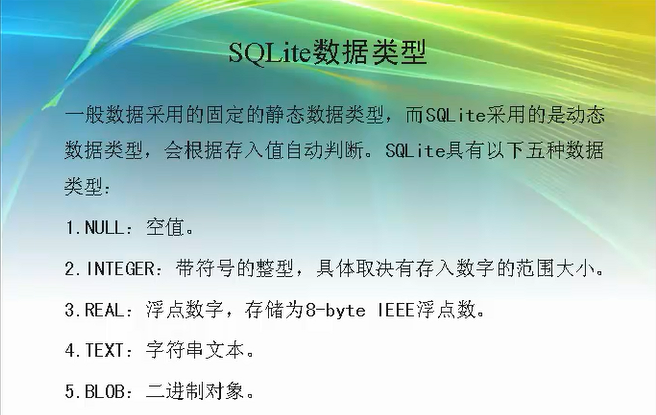
**public** Cursor select() {

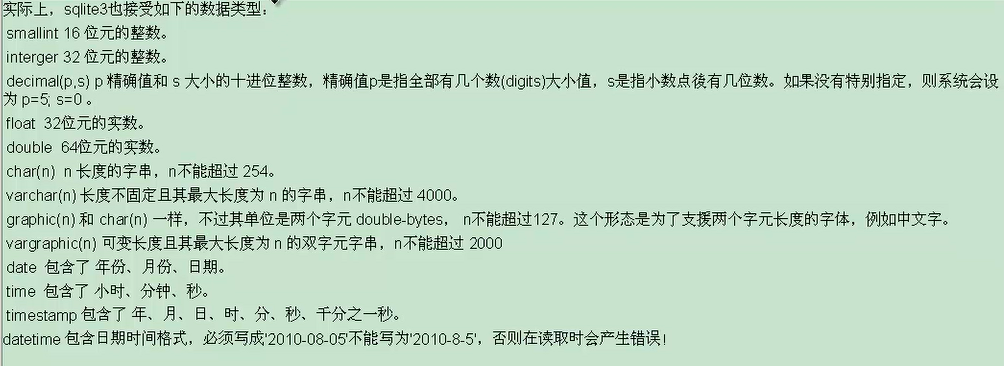
String sql = "select \* from user";

Cursor cursor = database.rawQuery(sql, **null**);

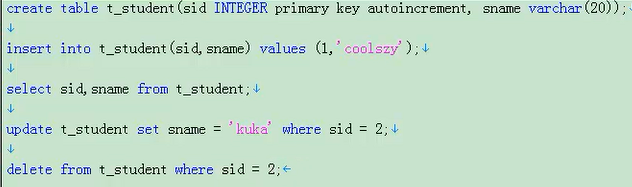
**return** cursor;

}



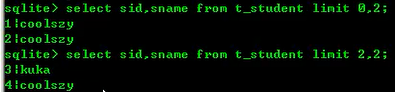


Sqlite基本语句





按id递减排序



Limit分页，第一个参数是起始位置，第二个参数是长度。

## Stringbuffer 的使用

Cursor cur = db.select();

StringBuffer sf = **new** StringBuffer();

//将游标指针指向第一行之前的位置。

cur.moveToFirst();

//循环判断，是否指向游标中的最后一行数据

**while** (!cur.isAfterLast()) {

//cur.getInt和rs.getString(0)，取得当前指针所指向的该行数据第一个字段的值。

sf.append(cur.getInt(0)).append(" : ")

.append(cur.getInt(1)).append(" : ").append(

cur.getString(2)).append("\n");

//将游标指针向下移动

cur.moveToNext();

}

Toast.*makeText*(AndroidTest.**this**, "search Table t\_user ok",

Toast.*LENGTH\_SHORT*).show();

list.setText(sf.toString());

}

## 39， googlemaps

正向编码 （由地名得到经纬度坐标）

Geocoder gc = **new** Geocoder(**this**, Locale.*getDefault*());

String address = "北京";

List<Address> locations = **null**;

**try** {

locations = gc.getFromLocationName(address, 3);

**if** (locations.size() > 0) {

Address a = locations.get(0);

**double** lat = a.getLatitude();

**double** lng = a.getLongitude();

tv.setText(lat + ":" + lng);

}

} **catch** (Exception e) {

e.printStackTrace();

}

反向编码：

**double** lng = 121.614682;

**double** lat = 38.914003;

Geocoder gc = **new** Geocoder(**this**, Locale.*getDefault*());

List<Address> addresses = **null**;

**try** {

addresses = gc.getFromLocation(lat, lng, 2);

StringBuilder builder = **new** StringBuilder();

Address a = addresses.get(0);

builder.append(a.getLocality()).append("\n");

builder.append(a.getCountryName());

builder.append(a.getPhone());

tv.setText(builder);

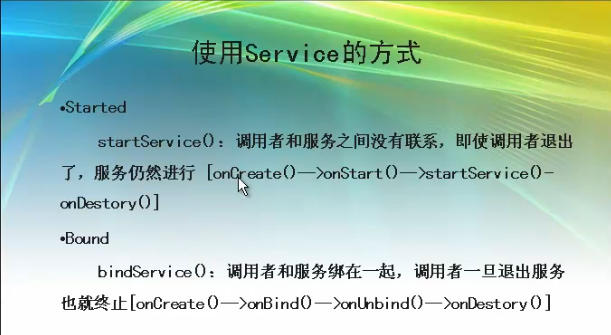
} **catch** (Exception e) {

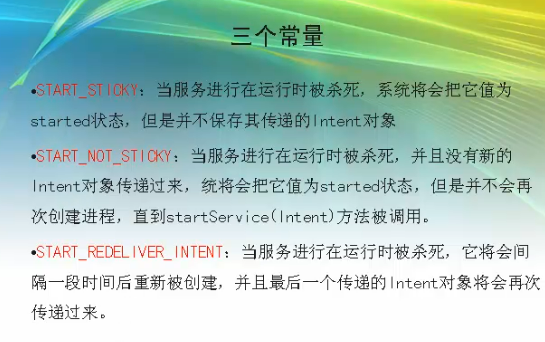
e.printStackTrace();

}

Mapview

## 40.Service





启动服务会调用 oncreate 、onstartcommand/onstart、

停止服务会调用 ondestory

启动服务后按键不会停止服务。

启动、停止、绑定、取消服务实现

连接服务：

**private** ServiceConnection coon = **new** ServiceConnection() {

@Override

**public** **void** onServiceDisconnected(ComponentName name) {

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

Log.*i*("bbb", "onServiceDisconnected");

}

@Override

**public** **void** onServiceConnected(ComponentName name, IBinder service) {

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

Log.*i*("bbb", "onServiceConnected");

bind = (Myservices.Mybind) service;

flag = **true**;

}

};

**private** OnClickListener listener = **new** View.OnClickListener() {

@Override

**public** **void** onClick(View v) {

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

Intent intent = **new** Intent();

intent.setAction("aaabbbccc");

**switch** (v.getId()) {

**case** R.id.*start*:

startService(intent);

**break**;

**case** R.id.*stop*:

stopService(intent);

**break**;

**case** R.id.*bind*:

bindService(intent, coon, Service.*BIND\_AUTO\_CREATE*);

**break**;

**case** R.id.*unbind*:

**if** (flag) {

unbindService(coon);

} **else** {

Toast.*makeText*(getApplicationContext(), "您还没有绑定任何服务",

Toast.*LENGTH\_SHORT*).show();

}

**break**;

**case** R.id.*button3*:

**if** (flag) {

Toast.*makeText*(getApplicationContext(), bind.getvalue(),

Toast.*LENGTH\_SHORT*).show();

} **else** {

Toast.*makeText*(getApplicationContext(), "您还没有绑定任何服务",

Toast.*LENGTH\_SHORT*).show();

}

**break**;

**default**:

**break**;

}

}

};

## 41,broadcast

先在manifest中注册广播接收者。设置action

<receiver android:name=*".Receiver1"* >

<intent-filter android:priority=*"300"*>

<action android:name=*"aaabbbccc"* />

</intent-filter>

</receiver>

<receiver android:name=*".Receiver2"* >

<intent-filter android:priority=*"500"*>

<action android:name=*"aaabbbccc"* />

</intent-filter>

</receiver>

<receiver android:name=*".Receiver3"* >

<intent-filter android:priority=*"100"*>

<action android:name=*"aaabbbccc"* />

</intent-filter>

</receiver>

Android：priority 是广播接收者的权限取值在-1000～1000

通过代码注册

MyReceiver mr = **new** MyReceiver();

//和intentFilter标签的作用一样

IntentFilter filter = **new** IntentFilter();//<intent-filter></intent-filter>

filter.addAction("com.ambow.android.MYACTION");//<intent-filter><action android:name="com.ambow.android.MYACTION"/></intent-filter>

registerReceiver(mr, filter);

取消注册

unregisterReceiver(mr);

发送广播 附带信息。

Intent intent = **new** Intent();

intent.setAction("aaabbbccc");

intent.putExtra("name", "jack");

sendOrderedBroadcast(intent, **null**);

sendBroadcast(intent);是所有人都能收到。

Sendorderebroadcast 如果第一个接收者把它终止掉的话。其它接收者就无法收到广播。

用sendorderebroadcast 发送广播时，第一个广播接收者还可以添加信息：

String name = intent.getStringExtra("name");

Log.*i*("bbbbbbbbbb", name);

// abortBroadcast();

Bundle b= **new** Bundle();

b.putString("age", "999");

setResultExtras(b);

abortbroadcast（）;是终止广播。

通过bundle 附加信息。

第二个接收者可以接收所有信息：

String name = intent.getStringExtra("name");

// String age = intent.getStringExtra("age");

Bundle bundle = getResultExtras(**true**);

String age = bundle.getString("age");

Log.*i*("aaaaaaaaaaa", name+age);

// abortBroadcast();

Getresultextras(true)是如果没有Bundle的话自动创建一个。

发送短信时广播：



把android.provider.Telephony.SMS\_RECEIVED设到action里。

同时要添加用户权限。

<uses-permission android:name=*"android.permission.RECEIVE\_SMS"*/>

<receiver android:name=*".Receiver1"* >

<intent-filter android:priority=*"300"*>

<action android:name=*"android.provider.Telephony.SMS\_RECEIVED"* />

</intent-filter>

</receiver>

接收短信内容

**if** ("android.provider.Telephony.SMS\_RECEIVED"

.equals(intent.getAction())) {

Bundle bundle = intent.getExtras();

StringBuffer stringBuffer = **new** StringBuffer();

**if** (bundle != **null**) {

Object[] objects = (Object[]) bundle.get("pdus");

SmsMessage[] messages = **new** SmsMessage[objects.length];

**for** (**int** i = 0; i < messages.length; i++) {

messages[i] = SmsMessage.*createFromPdu*((**byte**[]) objects[i]);

}

**for** (SmsMessage message : messages) {

stringBuffer.append("短信来源:");

stringBuffer.append(message.getDisplayOriginatingAddress())

.append("\n");

stringBuffer.append("短信内容：");

stringBuffer.append(message.getDisplayMessageBody());

}

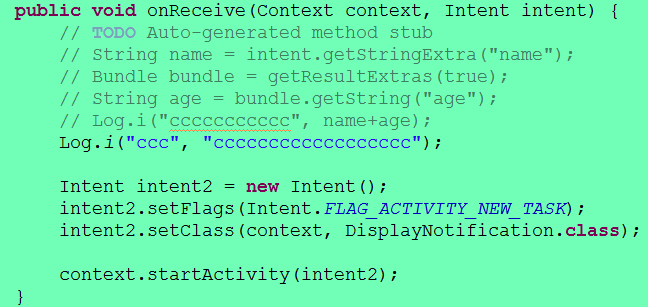
}

Toast.*makeText*(context, stringBuffer.toString(), Toast.*LENGTH\_LONG*).show();

}

在onreceive方法里继续跳转

Intent2.setflag 必须要有。



## 43,系统通知

**public** **class** AndroidTest **extends** Activity {

**private** Button sendBtn;

**private** Button cancelBtn;

**private** Notification n; //对话框，作用都是用来为用户提示信息

**private** NotificationManager nm;

**private** **int** ID = 1;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

sendBtn = (Button) findViewById(R.id.*button01*);

cancelBtn = (Button) findViewById(R.id.*button02*);

String service = *NOTIFICATION\_SERVICE*;

nm = (NotificationManager) getSystemService(service);

n = **new** Notification();

**int** icon = R.drawable.*icon*;//指定通知的图标

String tickerText = "Test Notification";//在系统的信息栏中的提示信息(概要信息)

**long** when = System.*currentTimeMillis*();//通知发送的时间

n.icon = icon;

n.tickerText = tickerText;

n.when = when;

sendBtn.setOnClickListener(sendListener);

cancelBtn.setOnClickListener(cancelListener);

}

**private** OnClickListener sendListener = **new** OnClickListener(){

@Override

**public** **void** onClick(View v) {

//单击通知跳转信息

Intent intent = **new** Intent(AndroidTest.**this**,AndroidTest.**class**);

//GPS中边界检测（边界警告），PendingIntent和Intent的作用一样，只不过不会马上进行代码的处理，只有达到某种条件才能触发。

PendingIntent pi = PendingIntent.*getActivity*(AndroidTest.**this**, 0, intent, 0);

//Intent跳转页面，马上跳转到某Activity

//PendingIntent跳转页面，暂时不动，达到条件了，跳转到某Activity

n.setLatestEventInfo(AndroidTest.**this**, "My Title", "My Content", pi);

nm.notify(ID,n);//通知系统警告

ID++;

}

};

**private** OnClickListener cancelListener = **new** OnClickListener(){

@Override

**public** **void** onClick(View v) {

nm.cancel(ID);

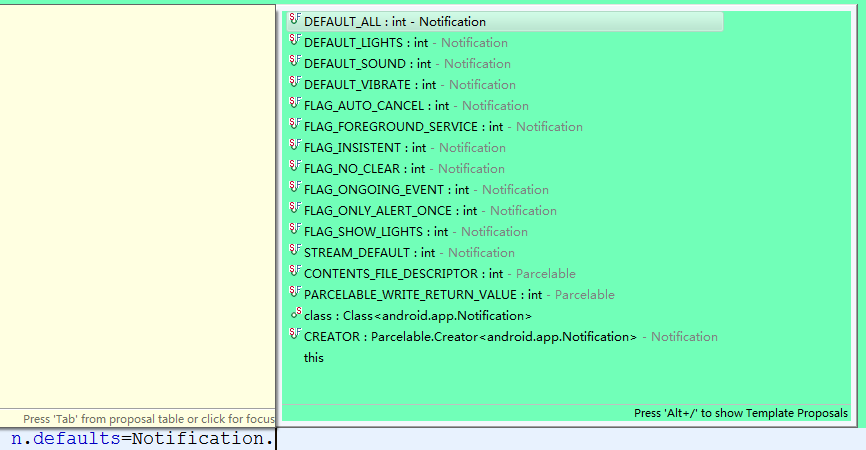
// nm.cancelAll();

}

};

}

补充：通知时的各种提示效果



## 44，AlarmManager 的使用

//AlarmManager.ELAPSED\_REALTIME表示闹钟在手机睡眠状态下不可用，该状态下闹钟使用相对时间（相对于系统启动开始），状态值为3；

//AlarmManager.ELAPSED\_REALTIME\_WAKEUP表示闹钟在睡眠状态下会唤醒系统并执行提示功能，该状态下闹钟也使用相对时间，状态值为2；

//AlarmManager.RTC表示闹钟在睡眠状态下不可用，该状态下闹钟使用绝对时间，即当前系统时间，状态值为1；

//AlarmManager.RTC\_WAKEUP表示闹钟在睡眠状态下会唤醒系统并执行提示功能，该状态下闹钟使用绝对时间，状态值为0；

//AlarmManager.POWER\_OFF\_WAKEUP表示闹钟在手机关机状态下也能正常进行提示功能，所以是5个状态中用的最多的状态之一，该状态下闹钟也是用绝对时间，

//状态值为4；不过本状态好像受SDK版本影响，某些版本并不支持

**public** **class** AndroidTest **extends** Activity {

**private** Button setBtn;

**private** Button cancelBtn;

**private** **static** **final** String *BC\_ACTION* = "com.ambow.android.BC\_ACTION";

/\*\* Called when the activity is first created. \*/

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

setBtn = (Button) findViewById(R.id.*button1*);

cancelBtn = (Button) findViewById(R.id.*button2*);

**final** AlarmManager am = (AlarmManager) getSystemService(*ALARM\_SERVICE*);

Intent intent = **new** Intent();

intent.setAction(*BC\_ACTION*);

intent.putExtra("msg", "吃饭了哈");

**final** PendingIntent pi = PendingIntent.*getBroadcast*(AndroidTest.**this**, 0, intent, 0);

**final** **long** time = System.*currentTimeMillis*();

setBtn.setOnClickListener(**new** OnClickListener(){

@Override

**public** **void** onClick(View v) {

//AlarmManager.RTC，表示是一个正常的定时器

//AlarmManager.RTC\_WAKEUP，表示除了有定时器的功能外，还会发出警报声（例如，响铃、震动）

am.setRepeating(AlarmManager.*RTC\_WAKEUP*, time, 8000, pi);

}

});

cancelBtn.setOnClickListener(**new** OnClickListener(){

@Override

**public** **void** onClick(View v) {

am.cancel(pi);

}

});

}

在myreceive里处理

**public** **class** MyReceiver **extends** BroadcastReceiver{

@Override

**public** **void** onReceive(Context context, Intent intent) {

String msg = intent.getStringExtra("msg");

Toast.*makeText*(context, msg, Toast.*LENGTH\_LONG*).show();

}

}

## 45，Calender的使用

**public** **class** AndroidTest **extends** Activity {

**private** Button button1;

**private** Button button2;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

button1 = (Button) findViewById(R.id.*button1*);

button2 = (Button) findViewById(R.id.*button2*);

button1.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View v) {

Intent intent = **new** Intent(AndroidTest.**this**, AndroidTest2.**class**);

PendingIntent pi = PendingIntent.*getActivity*(AndroidTest.**this**,

0, intent, 0);

Calendar calendar = Calendar.*getInstance*();

calendar.setTimeInMillis(System.*currentTimeMillis*());//当前系统时间

calendar.add(Calendar.*SECOND*, 5);

AlarmManager alarm = (AlarmManager) getSystemService(*ALARM\_SERVICE*);

alarm.set(AlarmManager.*RTC\_WAKEUP*, calendar.getTimeInMillis(), pi);

Toast.*makeText*(AndroidTest.**this**, "5秒后启动Activity2", Toast.*LENGTH\_SHORT*).show();

}

});

button2.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View v) {

Intent intent = **new** Intent(AndroidTest.**this**, AndroidTest2.**class**);

startActivity(intent);

}

});

}

}

## 46，TelephonyManager

**public** **class** TestActivity **extends** Activity {

**private** **static** **final** String *TAG* = "TestActivity";

**private** String[] telState = { "无活动", "响铃", "摘机" };

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

TelephonyManager tm = (TelephonyManager) **this**

.getSystemService(*TELEPHONY\_SERVICE*);

/\*

\* 电话状态： 1.tm.CALL\_STATE\_IDLE=0 无活动 2.tm.CALL\_STATE\_RINGING=1 响铃

\* 3.tm.CALL\_STATE\_OFFHOOK=2 摘机

\*/

Log.*i*(*TAG*, "电话状态为：" + telState[tm.getCallState()]);

/\*

\* 电话方位：

\*/

Log.*i*(*TAG*, "电话方位为：" + tm.getCellLocation());

/\*

\* 唯一的设备ID： GSM手机的 IMEI 和 CDMA手机的 MEID. Return null if device ID is not

\* available.

\*/

Log.*i*(*TAG*, "设备ID为：" + tm.getDeviceId());

/\*

\* 设备的软件版本号： 例如：the IMEI/SV(software version) for GSM phones. Return

\* null if the software version is not available.

\*/

Log.*i*(*TAG*, "设备的软件版本号为：" + tm.getDeviceSoftwareVersion());

/\*

\* 手机号： GSM手机的 MSISDN. Return null if it is unavailable.

\*/

Log.*i*(*TAG*, "手机号为：" + tm.getLine1Number());

/\*

\* 附近的电话的信息: 类型：List<NeighboringCellInfo>

\* 需要权限：android.Manifest.permission#ACCESS\_COARSE\_UPDATES

\*/

List<NeighboringCellInfo> list = tm.getNeighboringCellInfo();

**for** (NeighboringCellInfo nc : list) {

Log.*i*(*TAG*, "附近的电话的信息为：" + nc.describeContents());

}

/\*

\* 获取ISO标准的国家码，即国际长途区号。 注意：仅当用户已在网络注册后有效。 在CDMA网络中结果也许不可靠。

\*/

Log.*i*(*TAG*, "国际长途区号为：" + tm.getNetworkCountryIso());

/\*

\* MCC+MNC(mobile country code + mobile network code) 注意：仅当用户已在网络注册时有效。

\* 在CDMA网络中结果也许不可靠。

\*/

Log.*i*(*TAG*, "MCC+MNC为：" + tm.getNetworkOperator());

/\*

\* 按照字母次序的current registered operator(当前已注册的用户)的名字 注意：仅当用户已在网络注册时有效。

\* 在CDMA网络中结果也许不可靠。

\*/

Log.*i*(*TAG*, "名字为：" + tm.getNetworkOperatorName());

/\*

\* 当前使用的网络类型： 例如： NETWORK\_TYPE\_UNKNOWN 网络类型未知 0 NETWORK\_TYPE\_GPRS GPRS网络

\* 1 NETWORK\_TYPE\_EDGE EDGE网络 2 NETWORK\_TYPE\_UMTS UMTS网络 3

\* NETWORK\_TYPE\_HSDPA HSDPA网络 8 NETWORK\_TYPE\_HSUPA HSUPA网络 9

\* NETWORK\_TYPE\_HSPA HSPA网络 10 NETWORK\_TYPE\_CDMA CDMA网络,IS95A 或 IS95B. 4

\* NETWORK\_TYPE\_EVDO\_0 EVDO网络, revision 0. 5 NETWORK\_TYPE\_EVDO\_A EVDO网络,

\* revision A. 6 NETWORK\_TYPE\_1xRTT 1xRTT网络 7

\*/

Log.*i*(*TAG*, "网络类型为：" + tm.getNetworkType());

/\*

\* 手机类型： 例如： PHONE\_TYPE\_NONE 无信号 PHONE\_TYPE\_GSM GSM信号 PHONE\_TYPE\_CDMA

\* CDMA信号

\*/

Log.*i*(*TAG*,"手机类型为：" + tm.getPhoneType());

/\*

\* Returns the ISO country code equivalent for the SIM provider's

\* country code. 获取ISO国家码，相当于提供SIM卡的国家码。

\*/

Log.*i*(*TAG*,"国家码为：" + tm.getSimCountryIso());

/\*

\* Returns the MCC+MNC (mobile country code + mobile network code) of

\* the provider of the SIM. 5 or 6 decimal digits.

\* 获取SIM卡提供的移动国家码和移动网络码.5或6位的十进制数字. SIM卡的状态必须是

\* SIM\_STATE\_READY(使用getSimState()判断).

\*/

Log.*i*(*TAG*,"移动国家码和移动网络码为：" + tm.getSimOperator());

/\*

\* 服务商名称： 例如：中国移动、联通 SIM卡的状态必须是 SIM\_STATE\_READY(使用getSimState()判断).

\*/

Log.*i*(*TAG*,"服务商名称为：" + tm.getSimOperatorName());

/\*

\* SIM卡的序列号： 需要权限：READ\_PHONE\_STATE

\*/

Log.*i*(*TAG*,"SIM卡的序列号为：" + tm.getSimSerialNumber());

/\*

\* SIM的状态信息： SIM\_STATE\_UNKNOWN 未知状态 0 SIM\_STATE\_ABSENT 没插卡 1

\* SIM\_STATE\_PIN\_REQUIRED 锁定状态，需要用户的PIN码解锁 2 SIM\_STATE\_PUK\_REQUIRED

\* 锁定状态，需要用户的PUK码解锁 3 SIM\_STATE\_NETWORK\_LOCKED 锁定状态，需要网络的PIN码解锁 4

\* SIM\_STATE\_READY 就绪状态 5

\*/

Log.*i*(*TAG*,"SIM的状态信息为：" + tm.getSimState());

/\*

\* 唯一的用户ID： 例如：IMSI(国际移动用户识别码) for a GSM phone. 需要权限：READ\_PHONE\_STATE

\*/

Log.*i*(*TAG*,"唯一的用户ID为：" + tm.getSubscriberId());

/\*

\* 取得和语音邮件相关的标签，即为识别符 需要权限：READ\_PHONE\_STATE

\*/

Log.*i*(*TAG*,"取得和语音邮件相关的标签为：" + tm.getVoiceMailAlphaTag());

/\*

\* 获取语音邮件号码： 需要权限：READ\_PHONE\_STATE

\*/

Log.*i*(*TAG*,"获取语音邮件号码为：" + tm.getVoiceMailNumber());

/\*

\* ICC卡是否存在

\*/

Log.*i*(*TAG*,"ICC卡是否存在为：" + tm.hasIccCard());

/\*

\* 是否漫游: (在GSM用途下)

\*/

Log.*i*(*TAG*,"是否漫游：" + tm.isNetworkRoaming());

}

## 47，反射

**public** **void** get() **throws** Exception {

Class c = **null**;

c = Class.*forName*("java.lang.Long");

Class cs[] = {java.lang.String.**class** };

// Constructor cst1 = c.getConstructor(cs);

// System.out.println("通过参数获取指定Class对象的构造方法：");

// System.out.println(cst1.toString());

//

// Constructor[] csts = c.getConstructors();

// System.out.println("获取指定Class对象的所有构造方法：");

// for (int i = 0; i < csts.length; i++) {

// System.out.println(csts[i].toString());

// }

// Type types1[] = c.getGenericInterfaces();

// System.out.println("返回直接实现的接口：");

// for (int i = 0; i < types1.length; i++) {

// System.out.println(types1[i].toString());

// }

//

// Type type1 = c.getGenericSuperclass();

// System.out.println("返回直接超类：");

// System.out.println(type1.toString());

//

// Class cs1[] = c.getInterfaces();

// System.out.println("实现的接口：");

// for (int i = 0; i < cs1.length; i++) {

// System.out.println(cs1[i].toString());

// }

// Field fs1[] = c.getFields();

// System.out.println("类或接口的所有可访问公共字段：");

// for (int i = 0; i < fs1.length; i++) {

// System.out.println(fs1[i].toString());

// }

//

// Field fs2[] = c.getDeclaredFields();

// System.out.println("类或接口所声明的所有字段：");

// for (int i = 0; i < fs2.length; i++) {

// System.out.println(fs2[i].toString());

// }

//

// Method m1[] = c.getMethods();

// System.out.println("返回类所有的公共成员方法：");

// for (int i = 0; i < m1.length; i++) {

// System.out.println(m1[i].toString());

// }

//Method m3 = c.getMethod("longValue", cs);

// Method m2 = c.getMethod("longValue", new Class[] {});

// System.out.println("返回指定公共成员方法：");

// System.out.println(m2.toString());

}

方法调用 ：

**public** **class** Test2 {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

**try** {

Class c = Test2.**class**;

// Class c = Class.forName("com.ambow.android.Test2");

Object invokeTester = c.newInstance();

// 获取Test2类的add()方法

Method method1 = c.getMethod("add", new Class[] { int.class,

int.class });

// 调用invokeTester对象上的add()方法

Object result = method1.invoke(invokeTester, new Object[] {

new Integer(100), new Integer(200) });

System.*out*.println((Integer) result);

//获取Test2类的echo()方法

Method method2 = c.getMethod("echo", new Class[]{String.class});

//调用invokeTester对象的echo()方法

result = method2.invoke(invokeTester, new Object[]{"Hello"});

System.out.println((String) result);

//getName 是静态方法，不需要实例化对象调用。

Method method3 = c.getMethod("getName", null);

result = method3.invoke(null, null);

System.out.println((String)result);

} **catch** (InstantiationException e) {

e.printStackTrace();

} **catch** (IllegalAccessException e) {

e.printStackTrace();

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **int** add(**int** param1, **int** param2) {

**return** param1 + param2;

}

**public** String echo(String msg) {

**return** "echo: " + msg;

}

**public** **static** String getName(){

**return** "ford";

}

}

方法调用2：

**public** **class** Test3 {

**public** **double** x;

**public** **double** y;

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

**try**{

Class c = Test3.**class**;

Field xf = c.getField("x");

Field yf = c.getField("y");

Test3 t = **new** Test3();

//对象.方法，对象.属性

//方法.invoke(对象，参数)

//属性.get/set(对象)

//xf.get(t) = t.get(xf);

System.*out*.println("变更前x=" + xf.get(t));

xf.set(t, 1.1);

System.*out*.println("变更后x=" + xf.get(t));

System.*out*.println("变更前y=" + yf.get(t));

yf.set(t, 1.2);

System.*out*.println("变更后y=" + yf.get(t));

}**catch**(Exception e){

e.printStackTrace();

}

}

}

## 48，Sharedpreferences 存储数据

**public** **class** AndroidTest **extends** Activity {

**private** EditText txt;

**private** **static** **final** String *TEMP\_SMS* = "temp\_sms1";

SharedPreferences pre;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

txt = (EditText) findViewById(R.id.*txt*);

//利用getSharedPreferences方法，取得SharedPreferences对象

//方法的参数：1、文件的名称2、文件的模式

//MODE\_PRIVATE：代表文件是私有的，并且当我们生成Preferences File的时候，将内容全部替换

//MODE\_WORLD\_READABLE：允许其他应用程序对他进行读取。

//MODE\_WORLD\_WRITEABLE：允许其他应用程序对他进行写入。

//MODE\_APPEND：添加数据的时候，数据以追加的方法，放到文件内容的最后。

//得到当前的应用程序所在的文件夹下的临时文件

pre = getSharedPreferences(*TEMP\_SMS*, *MODE\_WORLD\_READABLE*);

//getString方法取得数据

//方法的参数：1、要取得的数据的key2、默认值（当没有找到符合条件的数据的话，会返回该默认值）

String content = pre.getString("sms\_content", **null**);//默认值，如果是String，null，如果是数字，0

txt.setText(content);

}

@Override

**protected** **void** onStop() {

**super**.onStop();

//Editor是用来编辑PreferencesFile数据的对象。

SharedPreferences.Editor editor = pre.edit();

//putString的作用是用来写入数据的。key-value

editor.putString("sms\_content", txt.getText().toString());

editor.putString("test", "test");

//commit提交，将写入的数据，真正的提交到物理文件中。提交数据，rollback

editor.commit();

}

}

从其它程序的包中取得数据，要取的数据必需要有可读的权限

**public** **class** AndroidTest **extends** Activity {

**private** EditText txt;

**private** **static** **final** String *TEMP\_SMS* = "temp\_sms1";

SharedPreferences pre;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

txt = (EditText) findViewById(R.id.*txt*);

//利用getSharedPreferences方法，取得SharedPreferences对象

//方法的参数：1、文件的名称2、文件的模式

//MODE\_PRIVATE：代表文件是私有的，并且当我们生成Preferences File的时候，将内容全部替换

//MODE\_WORLD\_READABLE：允许其他应用程序对他进行读取。

//MODE\_WORLD\_WRITEABLE：允许其他应用程序对他进行写入。

//MODE\_APPEND：添加数据的时候，数据以追加的方法，放到文件内容的最后。

//得到当前的应用程序所在的文件夹下的临时文件

pre = getSharedPreferences(*TEMP\_SMS*, *MODE\_WORLD\_READABLE*);

//getString方法取得数据

//方法的参数：1、要取得的数据的key2、默认值（当没有找到符合条件的数据的话，会返回该默认值）

String content = pre.getString("sms\_content", **null**);//默认值，如果是String，null，如果是数字，0

txt.setText(content);

}

@Override

**protected** **void** onStop() {

**super**.onStop();

//Editor是用来编辑PreferencesFile数据的对象。

SharedPreferences.Editor editor = pre.edit();

//putString的作用是用来写入数据的。key-value

editor.putString("sms\_content", txt.getText().toString());

editor.putString("test", "test");

//commit提交，将写入的数据，真正的提交到物理文件中。提交数据，rollback

editor.commit();

}

}

## 49，用输入输出流读写数据

**public** **class** AndroidTest **extends** Activity {

**private** **static** **final** String *FILE\_NAME* = "temp.txt";

**private** Button writeButton;

**private** Button readButton;

**private** EditText writeText;

**private** EditText readText;

/\*\* Called when the activity is first created. \*/

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

writeButton = (Button) findViewById(R.id.*writeButton*);

readButton = (Button) findViewById(R.id.*readButton*);

writeText = (EditText) findViewById(R.id.*writeText*);

readText = (EditText) findViewById(R.id.*readText*);

writeButton.setOnClickListener(**new** OnClickListener(){

**public** **void** onClick(View v) {

write(writeText.getText().toString());

}

});

readButton.setOnClickListener(**new** OnClickListener(){

**public** **void** onClick(View v) {

readText.setText(read());

}

});

}

**private** String read(){

**try**{

FileInputStream fis = openFileInput(*FILE\_NAME*);

**byte**[] buffer = **new** **byte**[fis.available()];

fis.read(buffer);

fis.close();

**return** **new** String(buffer);

}**catch**(Exception e){

e.printStackTrace();

}

**return** **null**;

}

**private** **void** write(String content){

**try**{

//openFileOutput参数1、写入的文件的名称2、写入文件的模式

FileOutputStream fos = openFileOutput(*FILE\_NAME*, *MODE\_APPEND*);

//write，写的操作

fos.write(content.getBytes());

fos.close();

}**catch**(Exception e){

e.printStackTrace();

}

}

}

## 50，解析 BitMap

InputStream is = getResources().openRawResource(R.drawable.*cat*);

Bitmap mBitmap = BitmapFactory.*decodeStream*(is);//编码

**public** **class** TestActivity **extends** Activity {

ImageView iv;

Button get1;

Button get2;

**static** **final** String *TAG* = "TestActivity";

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

iv = (ImageView) findViewById(R.id.*image*);

get1 = (Button) findViewById(R.id.*get1*);

get1.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View v) {

//得到当前应用程序下的assets文件夹下的资源

AssetManager am = getAssets();

**try** {

//得到assets目下的所有的资源

String[] images = am.list("");

**for**(**int** i = 0 ; i <images.length;i++){

Log.*i*("TestActivity", images[i]);

}

InputStream is = am.open(images[0]);

iv.setImageBitmap(BitmapFactory.*decodeStream*(is));

BitmapDrawable bd = (BitmapDrawable) iv.getDrawable();//保存在内存中的用于图片的对象

//下面的代码一定要注意，必须有，可以提升性能，减少内存消耗

//isRecycled方法用来判断当前某个ImageView所显示图片，是否在内存中存在

//recycle方法，用来释放ImageView所显示的图片，在内存中对应的对象

**if** (bd != **null** && !bd.getBitmap().isRecycled()) {

bd.getBitmap().recycle();

iv.setImageBitmap(BitmapFactory.*decodeStream*(is));

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

});

get2 = (Button) findViewById(R.id.*get2*);

get2.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View v) {

**try** {

//mnt/sdcard/icon.png

String path = Environment.*getExternalStorageDirectory*().getPath() + "/icon.png";

Bitmap bitmap = BitmapFactory.*decodeFile*(path);

iv.setImageBitmap(bitmap);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

}

}

## 51，使应用程序全屏

getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*, WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);

//隐去电池等图标和一切修饰部分（状态栏部分）

requestWindowFeature(Window.*FEATURE\_NO\_TITLE*);

// 隐去标题栏（程序的名字）

//android:theme=”@android:style/Theme.NoTitleBar” 隐去标题栏

//android:theme=”@android:style/Theme.NoTitleBar.Fullscreen” 隐去状态栏

代码的形式要写在setContentView前面。

// 全屏

requestWindowFeature(Window.*FEATURE\_NO\_TITLE*);

getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,

WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);

// 横屏显示

**this**.setRequestedOrientation(ActivityInfo.*SCREEN\_ORIENTATION\_LANDSCAPE*);

//设置半透明模式

getWindow().setFormat(PixelFormat.*TRANSLUCENT*);

## 52，SurfaceView

**public** **class** TestActivity **extends** Activity {

**private** **static** **final** String *TAG* = "TestActivity";

**private** **int** screenWidth;

**private** **int** screenHeight;

**private** Thread thread;

**private** Canvas canvas;

**private** SurfaceHolder sh;

**private** Paint paint;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

MyView mv = **new** MyView(**this**);

getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*, WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);

requestWindowFeature(Window.*FEATURE\_NO\_TITLE*);

setContentView(mv);

}

/\*

\* 被动更新。比如棋类，用View

\* 主动更新。比如一个人在一直跑动，用SurfaceView

\*

\* callback接口:继承SurfaceView类并实现SurfaceHolder.Callback接口就可以实现一个自定义的SurfaceView，

\* SurfaceHolder.Callback在底层的Surface状态发生变化的时候通知View。

\* SurfaceHolder类：用于控制surface的接口，它提供了控制surface 的大小，格式，上面的像素，即监视其改变的。

\*

\* SurfaceView的getHolder()函数可以获取SurfaceHolder对象，Surface 就在SurfaceHolder对象内。

\* 虽然Surface保存了当前窗口的像素数据，但是在使用过程中是不直接和Surface打交道的，

\* 由SurfaceHolder的Canvas lockCanvas()或Canvas lockCanvas()函数来获取Canvas对象，

\* 通过在Canvas上绘制内容来修改Surface中的数据。如果Surface不可编辑或则尚未创建调用该函数会返回null，

\* 在 unlockCanvas() 和 lockCanvas()中Surface的内容是不缓存的，所以需要完全重绘Surface的内容，

\* 为了提高效率只重绘变化的部分则可以调用lockCanvas(Rect rect)函数来指定一个rect区域，这样该区域外的内容会缓存起来。

\* 在调用lockCanvas函数获取Canvas后，SurfaceView会获取Surface的一个同步锁直到调用unlockCanvasAndPost(Canvas canvas)函数才释放该锁，

\* 这里的同步机制保证在Surface绘制过程中不会被改变（被摧毁、修改）。

\*/

**class** MyView **extends** SurfaceView **implements** Callback, Runnable {

**public** MyView(Context context) {

**super**(context);

paint = **new** Paint();

//消除锯齿

paint.setAntiAlias(**true**);

paint.setColor(Color.*RED*);

thread = **new** Thread(**this**);

sh = **this**.getHolder();

sh.addCallback(**this**);

}

**public** **void** paint() {

Log.*i*(*TAG*, "paint");

**try** {

canvas = sh.lockCanvas();

canvas.drawColor(Color.*WHITE*);

Random rw = **new** Random();

Random rh = **new** Random();

**int** pointX = rw.nextInt(screenWidth) + 1;

**int** pointY = rh.nextInt(screenHeight) + 1;

canvas.drawText("ambow", pointX, pointY, paint);

} **catch** (Exception ex) {

} **finally** {

**if** (canvas != **null**){

sh.unlockCanvasAndPost(canvas);

}

}

}

@Override

**public** **void** run() {

**while** (**true**) {

paint();

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

@Override

**public** **void** surfaceChanged(SurfaceHolder holder, **int** format, **int** width,

**int** height) {

// 当Surface的状态发生变化的时候会调用该函数，在surfaceCreated调用后该函数至少会被调用一次。

Log.*i*(*TAG*, "surfaceChanged");

}

@Override

**public** **void** surfaceCreated(SurfaceHolder holder) {

// 当Surface第一次创建后会立即调用该函数，程序可以在该函数中做些和绘制界面相关的初始化工作。

Log.*i*(*TAG*, "surfaceCreated");

screenWidth = getWidth();

screenHeight = getHeight();

thread.start();

}

@Override

**public** **void** surfaceDestroyed(SurfaceHolder holder) {

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

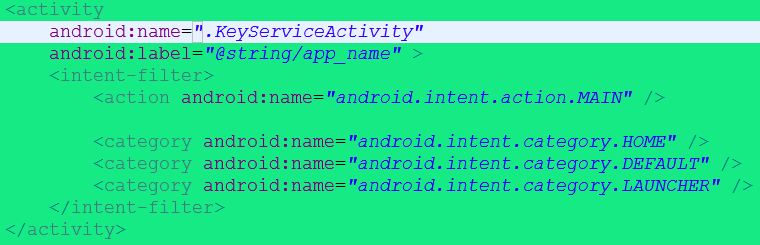
Log.*i*(*TAG*, "surfaceDestroyed");

}

}

}

## 把一个程序设成home app



## 横竖屏切换不回调oncreat

Mainfest下加入如下代码：

<activity

android:name=*".home.Home"*

android:configChanges=*"keyboardHidden|orientation"* >

</activity>

Ambow经纬度

