# TASK

**PreferenceActivity源码浅析：https://blog.csdn.net/yanbober/article/details/47954653**

# Preference

**case** com.android.internal.R.styleable.***Preference\_widgetLayout***:  
 **mWidgetLayoutResId** = a.getResourceId(attr, **mWidgetLayoutResId**);  
 **break**;

Android系统的Setting应用及我们市面上一些符合Android设计思想的应用的设置界面一般都会用它来实现，而且Google原生Android代码中大量的使用了Preference组件。

简单说，Preference组件其实就是Android常见UI组件与SharePreferences的组合封装实现。

# 初识Preference及Preference系

**另一种构建UI的方式——通过Preference去构建UI，而不是直接通过layout方式**（或许说得不够准确，因为Preference系也是间接通过Activity去展示的，只不过对于很多新人来说这种方式会和普通的有点不一样）

## Preference家族概述

如下图所示Preference作为该家族的终极基类，直接继承Object，其他子类Preference直接或者间接继承于Preference。他们的作用就是定义要显示的UI（其实作用类似于普通的Layout布局文件），特殊之处在于定义好了UI之后再在PreferenceActivity里以ListView的形式整体布局，每一个子Preference相当于是一个列表项。另外Preference还提供了一个SharedPreference用于保存/读取数据，以其key属性作为SharedPreference的键。还有一点Preference所存储的数据最后都会以xml文件格式的形式进行保存，而且其只能 保存一些基本格式的数据。例如string/boolean等等。该xml文件存放的位置在data/data/你应用的包名/shared\_prefs 文件夹下。

## 重要的家族成员

### Preference

#### 重要的回调接口和方法

| **接口（方法）** | **说明** |
| --- | --- |
| public static interface Preference.**OnPreferenceChangeListener** | 当对应的首选项的值改变时触发该接口的回调 |
| public static interface Preference.**OnPreferenceClickListener** | 当点击Prefrence时触发该接口的回调 |
| abstract boolean **onPreferenceChange**(Preference preference, Object newValue) | OnPreferenceChangeListener接口里对应的回调方法 |
| abstract boolean **onPreferenceClick**(Preference preference) | OnPreferenceClickListener接口对应的回调方法 |

简单来说就是当我们点击或者首选项值改变时候会分别触发这两个方法。

#### 重要的方法和属性

这些重要的公开方法或者接口都将被继承到其子类Preference下发挥重要的作用，所以又必要去眼熟下（我们都知道JavaBean中一般都是getter和setter成对出现的，为了节约篇幅就没有把所有的都列出来，还有父类的一些属性也没有全部列出，下同）

| **公共方法（属性）** | **说明** |
| --- | --- |
| **Preference**(Context context, AttributeSet attrs, int defStyle) |  |
| **Preference**(Context context, AttributeSet attrs) |  |
| android:key | key属性，可以唯一辨识同一容器内的Preference |
| android:title | 标题 |
| android:summary | Preference里的第二行的描述说明位于title下 |
| Context **getContext**() | 获取当前的上下文对象 |
| **SharedPreferences.Editor** getEditor() | 获取对应的SharedPreferences的Editor对象 |
| void **setFragment**() | 设置要显示的Fragment |
| void **setIcon**(Drawable icon)/**setIcon**(int iconResId) | 设置Preference的icon |
| void **setIntent**(Intent intent) | 当点击Preference时执行startActivity(intent)里的intent对象 |
| void **setOnPreferenceChangeListener**(Preference.OnPreferenceChangeListener listener) | 设置监听 |
| void**setOnPreferenceClickListener**(Preference.OnPreferenceClickListener listener) | 设置点击监听 |
| void **setSummary**(int summaryResId)/**setSummary**(CharSequence summary) | 设置summary |
| void **setKey**(String key) | 设置对应的key |
| void **setLayoutResource**(int layoutResId) | 设置Preference的布局 |
| protected void **onBindView**(View view) | 绑定数据 |
| protected View **onCreateView**(ViewGroup parent) | 建立Preference的布局类似Fragement里的 |
| protected void **onAttachedToActivity**() | 与附着的Activity对接时也是与Fragement里的类似 |
| void **notifyChanged**() | 当Preference里的值改变时候，可以手动调用这个方法去通知系统进行相关的更新，类似ListView里的 |

## PreferenceGroup

**PreferenceGroup**是直接继承于**Preference**，起到的是一个容器作用，类似于ViewGroup的功能，我们在日常开发中接触最多的是他的子类 **PreferenceCategory** 和**PreferenceScreen**，这两个直接继承PreferenceGroup所以在开发中起的作用也是容器，用于”安置“其他子Preference的容器。值得注意的是除了继承一些来自Preference及其他父类公共的方法还继承了一些公共属性

| **公共方法（属性）** | **说明** |
| --- | --- |
| **PreferenceGroup**(Context context, AttributeSet attrs, int defStyle) |  |
| **PreferenceGroup**(Context context, AttributeSet attrs) |  |
| void addItemFromInflater(Preference preference) | 当Preference被添加到当前容器时被调用 |
| Preference **findPreference**(CharSequence key) | 通过key属性找到对应的Preference，同一容器内key应该唯一 |
| int **getPreferenceCount**() | 获取容器内Preference的数量 |
| void **removeAll**() | 删除所有容器内的Preference |
| boolean **removePreference**(Preference preference) | 删除所有容器内的指定的Preference |
| boolean **onPrepareAddPreference**(Preference preference) | 添加Preference之前执行 |

## public final class PreferenceScreen

以上两个家族成员，我们在普通的开发中一般都是不会直接使用的，当人如果自定义Preference可能会涉及到，我们看到做多的应该是**PreferenceScreen**和**PreferenceCateGory**，首先按照官网描述，**PreferenceScreen**在一个Preference的层级结构中是作为top-level顶级层次的，类似于普通布局中的各种Layout，LinearLayout、RelativeLayout等等，他继承于PreferenceGroup，所以同样起的容器的作用。同样的我们除了可以通过**xml方式构造PreferenceScreen**还可以通过**createPreferenceScreen(Context)**(后面文章再说具体用法)。

### PreferenceScreen

* **作为一个PreferenceAcitivity的基本布局的根容器**：和其他根Layout一样，是不可见的容器，只是把容器内的Preference呈现出来
* **嵌套插入在其他PreferenceScreen容器里**

Fd

<PreferenceScreen

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

android:key="first\_preferencescreen">

<CheckBoxPreference

android:key="wifi enabled"

android:title="WiFi" />

<PreferenceScreen

android:key="second\_preferencescreen"

android:title="WiFi settings">

<CheckBoxPreference

android:key="prefer wifi"

android:title="Prefer WiFi" />

... other preferences here ...

</PreferenceScreen>

</PreferenceScreen>

### 重要的方法和属性

| **公共方法（属性）** | **说明** |
| --- | --- |
| void bind(ListView listView) | Binds a ListView to the preferences contained in this PreferenceScreen via getRootAdapter(). |
| Dialog **getDialog**() | Used to get a handle to the dialog. |
| ListAdapter **getRootAdapter**() | Returns an adapter that can be attached to a PreferenceActivity or PreferenceFragment to show the preferences contained in this PreferenceScreen. |
| void **onItemClick**(AdapterView parent, View view, int position, long id) |  |
| void **onDismiss**(DialogInterface dialog) | This method will be invoked when the dialog is dismissed. |
| protected void **onClick**() | Processes a click on the preference. |
| protected ListAdapter **onCreateRootAdapter**() | Creates the root adapter. |
| Parcelable **onSaveInstanceState**() | Hook allowing a Preference to generate a representation of its internal state that can later be used to create a new instance with that same state. |

## PreferenceCategory

**PreferenceCategory**也是继承于PreferenceGroup，所以他也具有容器的功能，不过一般不用做顶级容器，可用于二级容器嵌套在PreferenceScreen里提供分组的作用，和数据库SQL中的group by差不多。

# DialogPreference（abstract）

**DialogPreference**直接继承自**Preference**，它的独特之处在于它是**基于Dialog的**，也就是说但我们点击对应的DialogPreference系时是**以Dialog形式展现**的，又因它是一个abstract抽象类，所以更多的时候我们直接使用它的子类：**EditTextPreference**、**ListPreference**和**MultiSelectListPreference**

| **DialogPreference系公共方法（属性）** | **说明** |
| --- | --- |
| android:**dialogIcon** | 对话框的icon |
| android:**dialogLayout** | dialog 的contentView 布局 |
| android:**dialogMessage** | 对话框的内容 |
| android:**dialogTitle** | 对话框的标题 |
| android:**negativeButtonText** | 对话框里的按钮1名称 |
| android:**positiveButtonText** | 对话框里的按钮2名称 |
| **DialogPreference**(Context context, AttributeSet attrs, int defStyle) |  |
| **DialogPreference**(Context context, AttributeSet attrs) |  |
| void **onActivityDestroy**() | 当Activity destroy时触发 |
| void **onClick**(DialogInterface dialog, int which) | 对话框统一的回调方法，可以通过which来判断点击的是哪个按钮 |
| void **onDismiss**(DialogInterface dialog) | 但对话框消失时，当对话框消失后执行，其实就是DialogInterface.OnDismissListener的回调方法 |
| 一系列对应xml属性的setter和getter方法略… |  |
| protected void **onBindDialogView**(View view) | 绑定数据 |
| protected void **onClick**() | 处理onClick事件，当对话框里的按钮被点击时，可以通过which来判断点击的按钮，其实就是DialogInterface.OnClickListener的回调方法 |
| protected View **onCreateDialogView**() | 建立Dialog的contentView |
| protected void **onDialogClosed**(boolean positiveResult) | 当dialog 消失时，常被用于保存数据到SharedPreferences |
| protected void **onPrepareDialogBuilder**(AlertDialog.Builder builder) | 当Preference被点击前，dialog显示之前触发 |
| protected void **onRestoreInstanceState**(Parcelable state) | 恢复在onSaveInstanceState保存的数据 |
| protected Parcelable **onSaveInstanceState**() | 保存状态 |
| protected void **showDialog**(Bundle state) | 显示关联的dialog |

## DialogPreference系独有属性

应用与管理

### DialogPreference UI 原理

DialogPreference

**final** TypedArray a = context.obtainStyledAttributes(attrs,  
 com.android.internal.R.styleable.***DialogPreference***, defStyleAttr, defStyleRes);

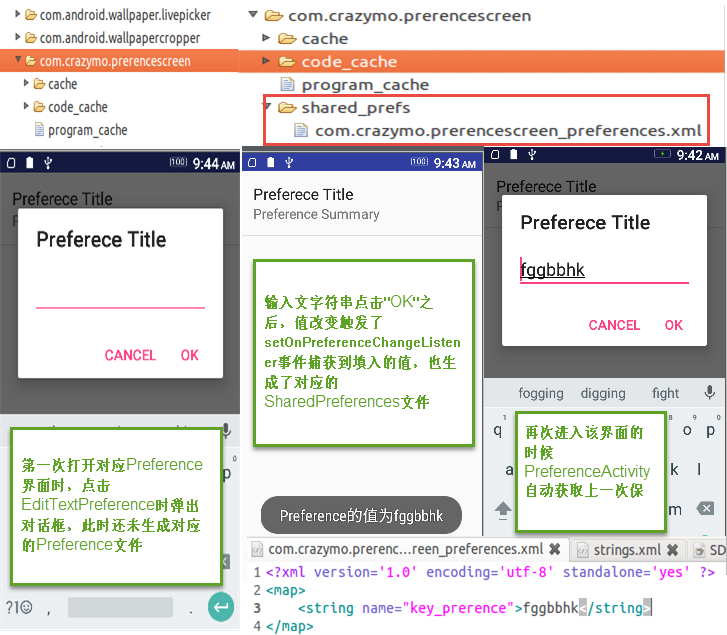
**mDialogLayoutResId** = a.getResourceId(com.android.internal.R.styleable.***DialogPreference\_dialogLayout***,  
 **mDialogLayoutResId**);

**public** DialogPreference(Context context, AttributeSet attrs) {  
 **this**(context, attrs, com.android.internal.R.attr.***dialogPreferenceStyle***);  
}

**protected** View onCreateDialogView() {  
 **if** (**mDialogLayoutResId** == 0) {  
 **return null**;  
 }  
   
 LayoutInflater inflater = LayoutInflater.*from*(**mBuilder**.getContext());  
 **return** inflater.inflate(**mDialogLayoutResId**, **null**);  
}

## EditTextPreference

| **EditTextPreference新增方法** | **说明** |
| --- | --- |
| EditText getEditText() | 获取显示在Dialog上的EditText |
| String getText() | 获取对应SharedPreferences保存的值 |



## ListPreference

特别之处在于SharedPreferences存储的是**android:entryValues的值（即android:entryValues与key一一对应形成键值对），而显示到界面的是android:entries则与android:entryValues一一对应**

<ListPreference

android:icon="@mipmap/ic\_launcher"

android:title="ListPreference"

android:summary="ListPreference summary"

android:dialogIcon="@mipmap/ic\_blue\_launcher"

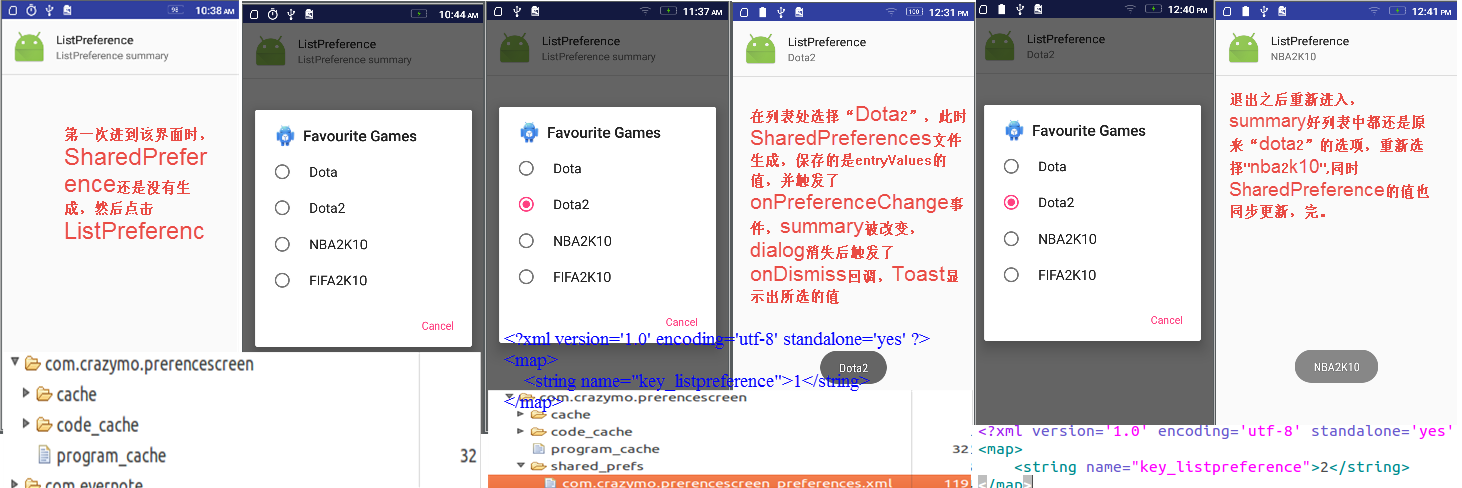
android:dialogTitle="Favourite Games"

android:key="key\_listpreference"

android:entries="@array/game"

android:entryValues="@array/game\_index"

/>



### ListPreference的特有方法

| **ListPreference新增方法** | **说明** |
| --- | --- |
| int findIndexOfValue(String value) | 返回获取ListPreference中的实体内容的下标值 |
| CharSequence[] getEntries() | 返回当前Preference设置的entries集合(xml配置的字符串数据源) |
| CharSequence getEntry() | 返回当前选中的entries值 |
| CharSequence[] getEntryValues() | 返回当前的entries对应的value集合(xml配置的字符串数据源) |
| CharSequence getSummary() | 获取当前summary |
| String getValue() | 获取key值 |

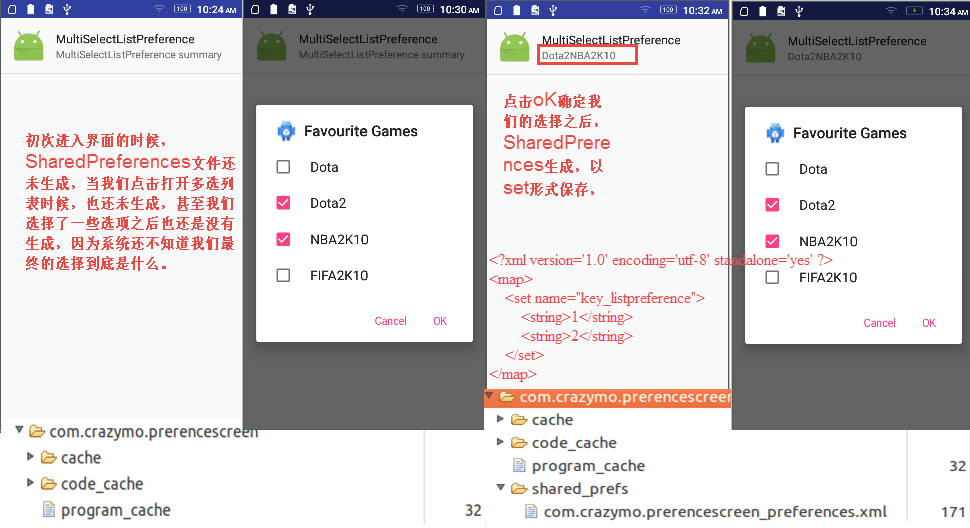
## MultiSelectListPreference

MultiSelectListPreference和ListPreference都是列表形式的DialogPreference，不同的是ListPreference提供单选列表，而在Android3.0之后可以通过MultiSelectListPreference来实现多选，所以他们的属性和方法都几乎是相同的，当然也包括使用基本语法，但是要注意一点由于要支持多选，所以**MultiSelectListPreference的SharedPreferences保存的是集合set**（我们应该有这个概念集合就是用于保存多个对象的），所以读取数据的时候处理稍显麻烦。

#### MultiSelectListPreference新增的方法

| **MultiSelectListPreference新增方法** | **说明** |
| --- | --- |
| **int findIndexOfValue**(String value) | 返回获取ListPreference中的实体内容的下标值 |
| **CharSequence[] getEntries**() | 返回当前Preference 设置的entries集合(xml配置的字符串数据源) |
| **CharSequence[] getEntryValues**() | 返回当前的entries对应的value集合(xml配置的字符串数据源) |
| **Set getValues**() | 返回当前选中的values，SharedPreference的 |
| void setEntries(int entriesResId) |  |
| void setEntries(CharSequence[] entries) |  |
| void setEntryValues(CharSequence[] entryValues) |  |
| void setEntryValues(int entryValuesResId) |  |
| void setValues(Set values) |  |

https://blog.csdn.net/zhangyongfeiyong/article/details/79157552



## EditTextPreferenc

作为DialogPreference的直接子类，这些子类Preference肯定拥有了DialogPreference的一切共性和特点，第一个肯定还是以Dialog的形式弹出，区别在于弹出的界面构成和其他的特性，这些在我们以后在开发中经常使用到，暂不详讲。

# ListPreference

**public** ListPreference(Context context, AttributeSet attrs) {  
 **this**(context, attrs, com.android.internal.R.attr.***dialogPreferenceStyle***);  
}

***dialogPreferenceStyle***

ndroid.support.v7.preference;

**public** ListPreference(Context context, AttributeSet attrs) {  
 **this**(context, attrs, TypedArrayUtils.*getAttr*(context, R.attr.***dialogPreferenceStyle***,  
 android.R.attr.***dialogPreferenceStyle***));  
}

## Theme.xml

<**item name="dialogPreferenceStyle"**>@style/Preference.DialogPreference</**item**>

### Style.xml

<**style name="Preference.DialogPreference"**>  
 <**item name="positiveButtonText"**>@string/ok</**item**>  
 <**item name="negativeButtonText"**>@string/cancel</**item**>  
</**style**>

*<!-- Preference Styles -->*<**style name="Preference"**>  
 <**item name="layout"**>@layout/preference</**item**>  
</**style**>

## DeviceDefault

<**item name="dialogPreferenceStyle"**>@style/Preference.DeviceDefault.DialogPreference</**item**>

<**style name="Preference.DeviceDefault.DialogPreference" parent="Preference.Material.DialogPreference"**/>

### Style- Material.xml

<**style name="Preference.Material.DialogPreference"**>  
 <**item name="positiveButtonText"**>@string/ok</**item**>  
 <**item name="negativeButtonText"**>@string/cancel</**item**>  
</**style**>

<**style name="Preference.Material"**>  
 <**item name="layout"**>@layout/preference\_material</**item**>  
</**style**>

### layout/preference\_material

</**RelativeLayout**>

<**RelativeLayout**

<**TextView android:id="@+id/title"  
 android:textAppearance="?attr/textAppearanceListItem"  
 android:ellipsize="marquee"** />  
<**TextView android:id="@+id/summary"  
 android:textAppearance="?attr/textAppearanceListItemSecondary"  
 android:textColor="?attr/textColorSecondary"  
 android:ellipsize="end"** />

*<!-- Preference should place its actual preference widget here. -->*<**LinearLayout android:id="@+id/widget\_frame"  
 android:orientation="vertical"** />

## Support思路！！！！！！！！！！

**Preference.DialogPreference.Material**

<**style name="Preference.DialogPreference.Material"**>  
 <**item name="android:layout"**>@layout/preference\_material</**item**>  
</**style**>

**android:background="?android:attr/selectableItemBackground"**>

<**TextView android:id="@android:id/title  
 android:textAppearance="@style/Preference\_TextAppearanceMaterialSubhead"** />

<**TextView android:id="@android:id/summary"  
 android:textAppearance="?android:attr/textAppearanceSmall"  
 android:textColor="?android:attr/textColorSecondary"** />

*<!-- Preference should place its actual preference widget here. -->*<**LinearLayout android:id="@android:id/widget\_frame"** />

### preference\_material

## 解决办法

**Rm500.Theme.Settings**

<**item name="android:dialogPreferenceStyle"**>@style/Rm500.Preference.DialogPreferenceStyle</**item**>

没有干活，属性值舍得不对吧

<**style name="Rm500.Preference.DialogPreferenceStyle" parent="@style/Preference.DialogPreference.Material"**>  
 <**item name="android:colorBackground"**>@color/test\_green\_00FF00</**item**>  
 <**item name="android:windowBackground"**>@color/test\_green\_00FF00</**item**>  
 <**item name="android:background"**>@color/test\_green\_00FF00</**item**>  
 <**item name="android:colorPrimary"**>@color/test\_green\_00FF00</**item**> *<!-- actionBar background3 ok-->*</**style**>

# TwoStatePreference

**TwoStatePreference**和**DialogPreference**同级，都是直接继承自Preference。**TwoStatePreference**如字面意思，是基于两种可选状态的首选项基类，在SharedPreferences里通过维护一个布尔值来设置当前状态的启用和禁止的一种Preference。

Android也为我们提供了两种：CheckBox选择项**CheckBoxPreference**和 开关选择**SwitchPreference**供我们直接使用。

## TwoStatePreference的共性

| **TwoStatePreference系独有方法** | **说明** |
| --- | --- |
| **CharSequence getSummaryOff**() | 获取unchecked状态下summary的值 |
| **CharSequence getSummaryOn**() | 获取checked状态下summary的值 |
| boolean **isChecked**() | 获取是否checked |
| void **setChecked**(boolean checked) | 设置checked状态 |
| void **setSummaryOff** | 设置unchecked下summary的值，可以通过字符串id和字符串 |
| void **setSummaryOn** | 设置checked下summary的值，可以通过字符串id和字符串 |

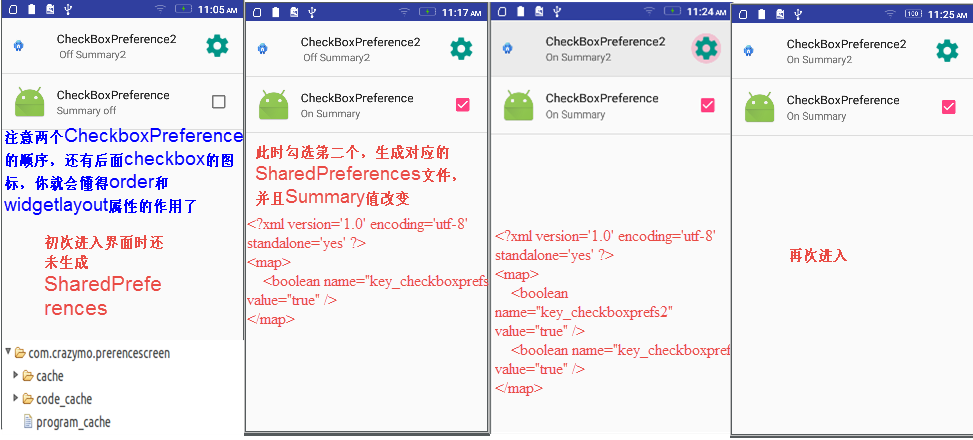
| **TwoStatePreference系公共方法** | **说明** |
| --- | --- |
| **TwoStatePreference**(Context context, AttributeSet attrs, int defStyle) |  |
| **TwoStatePreference**(Context context, AttributeSet attrs) |  |
| **TwoStatePreference**(Context context) |  |
| CharSequence **getSummaryOff**() | 获取当未check时显示的summary |
| CharSequence **getSummaryOn**() | 获取checked时显示的summary |
| void **setSummaryOff**(CharSequence summary)/(int summaryResId) |  |
| void **setSummaryOn**(CharSequence summary)/(int summaryResId) |  |
| boolean **isChecked**() |  |
| void **setChecked**(boolean checked) |  |
| protected void **onClick**() | 处理onclick |
| Object **onGetDefaultValue**(TypedArray a, int index) | 当Preference的布局文件被映射和需要读取默认值时被调用 |
| protected void **onRestoreInstanceState**(Parcelable state) |  |
| protected Parcelable **onSaveInstanceState**() |  |
| protected void **onSetInitialValue**(boolean restoreValue, Object defaultValue) | 设置首选项的初值 |

## CheckBoxPreference

CheckBoxPreference新增的属性

| **CheckBoxPreference新增属性** | **说明** |
| --- | --- |
| android:summaryOff | uncheck时的summary |
| android:summaryOn | check时的summary |

每次点击的时候**先执行onChange再执行onClick**：

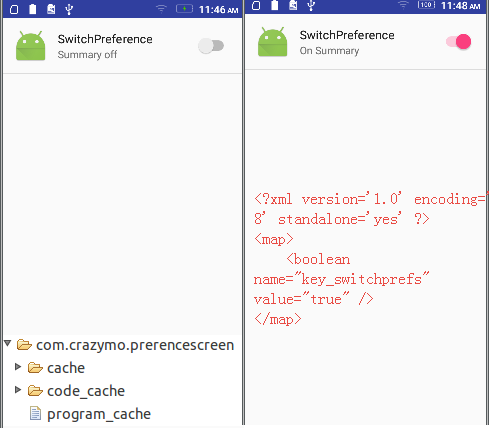


## SwitchPreference

### SwitchPreference新增属性

| **CheckBoxPreference新增属性** | **说明** |
| --- | --- |
| **android:summaryOff** | uncheck时的summary |
| **android:summaryOn** | check时的summary |
| **android:switchTextOff** |  |
| **android:switchTextOn** |  |

每次点击的时候**先执行onChange再执行onClick**：



# PreferenceActivity

**PreferenceActivity**是一个抽象类，**继承于ListActivity**，以列表形式视图来展现界面,加载的整个View也是基于ListActivity中那个ListView的，其最主要的优势在于添加Preference后可让其状态持久化储存（通过SharedPreferences，一般存储在**/data/data//shared\_prefs**文件夹下的默认名为**“app package name”+”\_preferences.xml”**的文件里）

## 初始化

PreferenceActivity其实和普通的Activity本质上来说区别不大，只不过多了些自动去读取SharedPrefrences的值来更新界面和其他一些逻辑，所以初始化本质上来说并无很大的区别，但是与普通Activity的layout不同，**PreferenceActivity的layout我们可以理解成为两个部分：其他View和一个id为android.R.id.list的ListView**，那么我们可以理解成为当我们在onCreate方法里**先调用setContentView完成整个Activity的View的构建**（layout文件里必须包含id为android.R.id.list的listView，否则会报E/AndroidRuntime: Caused by: java.lang.RuntimeException: Your content must have a ListView whose id attribute is ‘android.R.id.list’），再**调用addPreferencesFromResource来完成Preference界面的构建**；当然也可以**只调用addPreferencesFromResource**方法。

### 继承PreferenceActivity实现具体业务类，重写相关生命周期方法

Dgh

public class MainActivity extends PreferenceActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

addPreferencesFromResource(R.xml.demo\_preference);

}

...

}

### 通过****addPreferencesFromResource****（xml资源id）加载静态xml资源文件 或者 完全通过代码构造对象再动态添加

Sdf

在**res文件夹下新建xml文件夹，再在xml文件中新建对应的xml资源**，xml资源类似我们在使用普通Activity时的layout文件，PreferenceActivity独特之处在于并不是使用普通的layout文件，而是使用**res下xml文件夹下的xml资源文件**

res/xml/demo\_preference.xml

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

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

<CheckBoxPreference

android:key="key\_checkbox\_preference"

android:summary="Some summay for CheckBoxPreference"

android:title="The Title Of CheckBoxPreference" />

</PreferenceScreen>

* 1

也可通过代码去构造对象，添加容器之后再调用setPreferenceScreen（PreferenceScreen preferenceScreen）把容器对象设置到Activity上

private void createPreference(){

PreferenceScreen preferenceScreen = this.getPreferenceManager().createPreferenceScreen(this);//先构建PreferenceScreen对象得到一个布局容器

this.setPreferenceScreen(preferenceScreen);//设置容器

CheckBoxPreference checkBoxPreference=new CheckBoxPreference(this);//构建一个子Preference，待添加到容器中

checkBoxPreference.setKey(CHECKBOXPRERENCE\_KEY);//设置key

checkBoxPreference.setTitle("The Title Of CheckBoxPreference");//设置title

checkBoxPreference.setSummary("Some summay for CheckBoxPreference");

preferenceScreen.addPreference(checkBoxPreference);//添加到容器中

}

## 使用

1. 构造对象：java**对应的构造方法，**Xml构造
2. PreferenceActivity的初始化
3. 3.再接着根据业务设置相关监听

### 构造Preference容器:Preference对象

PreferenceScreen只能作为top-level节点

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

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

android:key="using\_categories\_in\_root\_screen"

android:summary="Using Preference Categories"

android:title="Categories">

<Preference

android:key="key\_prerence"

android:title="Preferece"

android:summary="Preference Demo"

/>

</PreferenceScreen>

### 初始化PreferenceActivity和设置相关监听

preference=findPreference(PREFERENCE\_KEY);

#### 设置Preference点击监听

preference.setOnPreferenceClickListener(new Preference.OnPreferenceClickListener() {

@Override

public boolean onPreferenceClick(Preference preference) {

//当接收到Click事件之后触发

return true;

}

});

#### 设置Preference对应的SharedPrefernces值监听

preference.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {

@Override

public boolean onPreferenceChange(Preference preference, Object newValue) {

//如果值改变了我们可以通过监听这个事件来处理

return true;

}

});

### Preference的常用xml属性

Df

| **xml属性** | **说明** |
| --- | --- |
| **android:icon** | Preference的icon |
| **android:summary** | 副标题、说明（小字体显示） |
| **android:title** | 标题 |
| **android:key** | 对应的SharedPrefrences的key值 |
| **android:layout** | PreferenceActivity的布局文件即如果设置了这个属性则会覆盖原来的UI |
| **android:fragment** | 应用在PreferenceActivity中时，当用户点击这个Preference时，启动一个新的Fragment |
| android:enabled | 设置Preference是否可用，false阴影状态不可操作 |
| android:order | Preference的排序,整数类型如”100” |
| android:persistent | true时，系统会帮助我们去保存该设置，即使重启后依然能记忆之前的设置，这也是所谓的持久化 。这里 将 android:persistent设置为False，表明不需要让系统去做持久化，开发者系统通过自己的方式去实现持久化。 |
| android:selectable | Preference是否可选，false可以点击但无响应 |
| android:shouldDisableView | 当View disabled的时候是否Preference也一样Disabled. |
| android:widgetLayout | 自定义Preference用到给子Preference定义布局。 |

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

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

android:title="PreferenceScreen">

<PreferenceCategory

android:key="key\_prerence"

android:summary="Preference Categories Summary"

android:title="Preference Categories">

<CheckBoxPreference

android:icon="@mipmap/ic\_launcher"

android:key="key\_chkpreference"

android:title="CheckBoxPreferenceCheckBoxPreferenceCheckBoxPreferenceCheckBoxPreference"

android:summary="CheckboxPreference summary"/>

<Preference

android:title="Preference"

android:key="key\_pre"

android:icon="@mipmap/ic\_red\_launcher"

android:layout="@layout/activity\_main"

/>

<EditTextPreference

android:icon="@mipmap/ic\_blue\_launcher"

android:key="key\_edtkpreference"

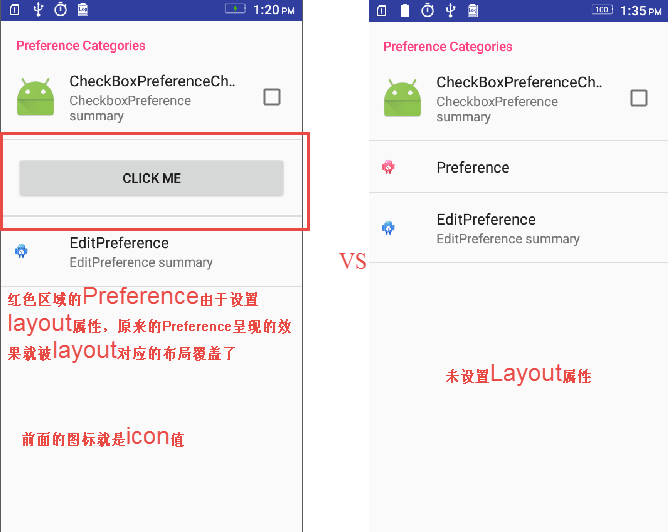
android:title="EditPreference"

android:summary="EditPreference summary"

/>

</PreferenceCategory>

</PreferenceScreen>



### Preference的xml文件中的常用标签****intent****、****extra****

Sdf

在xml文件的Preference标签中，我们可以添加intent来为我们快速实现一种意图，比如说快速打开一个网页，或者快速启动一个Activity等等，还可以使用**extra**给**intent**标签加参数来传递参数(再通过**getIntent().getStringExtra(“key”)**来获取)。

点击这个Preference则会自动去调用浏览器打开[**http://www.hao123.com**](http://www.hao123.com/)网页

<Preference

android:title="Click me open the web"

android:key="key\_pref\_intent"

>

<intent android:action="android.intent.action.VIEW"

android:data="http://www.hao123.com"/>

<!--可以通过<extra>传附加信息getIntent().getStringExtra("reused\_key") -->

<extra

android:name="key"

android:value="value"/>

</Preference>

启动指定类

<Preference

android:title="PREFERENCE TITLE"

>

<!-- android:targetPackage是应用程序的包名，而android:targetClass的路径在子包下的类 -->

<!-- android:targetPackage设置为子包，运行时则找不到Activity -->

<intent

android:action="ACTION\_A\_INTENT"

android:targetPackage="com.crazy.training"

android:targetClass="com.crazy.training.ui.MainActivity">

</intent>

</Preference>

# PreferenceScreen和PreferenceCategory

Asdf

PreferenceScreen和PreferenceCategory没有新增的属性，所有属性全部继承自Preference。其中PreferenceScreen作为顶级容器，PreferenceCategory作为次级容器（类似于SQL Group by功能暂且这么理解吧），虽然他们也是可以单独使用的，但是并**不能响应onPreferenceClick**和**onPreferenceChange**事件。

布局和MainActivity的代码依然很简单和前面类似

# final PreferenceScreen

Android系统为设置界面的UI提供了一系列的接口，设置界面的部分和Activity是分离的，会有一个PreferenceScreen的对象  
是根目录，在其中会包含CheckBoxPreference EditTextPreference ListPreference PreferenceCategory RingtonePreference

设置的界面是使用Preference而不是View来创建的，需要特殊的Activity或者Fragment的子类来显示

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

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

android:title="PreferenceScreen">

<PreferenceCategory

android:key="key\_prerence"

android:summary="Preference Categories Summary"

android:title="Preference Categories">

</PreferenceCategory>

</PreferenceScreen>



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

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

android:title="PreferenceScreen">

<PreferenceCategory

android:key="key\_prerence"

android:summary="Preference Categories Summary"

android:title="Preference Categories">

<CheckBoxPreference

android:icon="@mipmap/ic\_launcher"

android:key="key\_chkpreference"

android:title="CheckBoxPreference"

android:summary="CheckboxPreference summary"/>

<EditTextPreference

android:icon="@mipmap/ic\_blue\_launcher"

android:key="key\_edtkpreference"

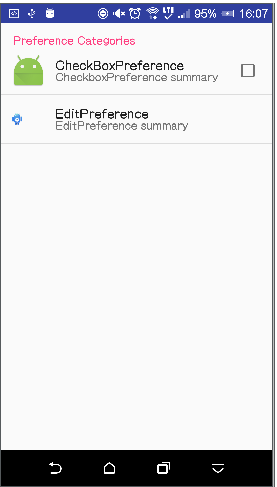
android:title="EditPreference"

android:summary="EditPreference summary"

/>

</PreferenceCategory>

</PreferenceScreen>



# RingtonePreference的应用

RingtonePreference起作用就是供我们选择系统铃声的，除了Preference共有的属性外还新增了自己的几个独特属性。

| **新增属性** | **说明** |
| --- | --- |
| android:ringtoneType | 铃声类型：ringtone、notification、all、alarm  ringtone/notification/alarm/all |
| android:showDefault | 布尔值是否显示默认铃声 |
| android:showSilent | 布尔值是否显示静音 |

应用也很简单，与Preference大同小异（注意看图）

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

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

android:key="using\_categories\_in\_root\_screen"

android:summary="Using Preference Categories"

android:title="Categories">

<RingtonePreference

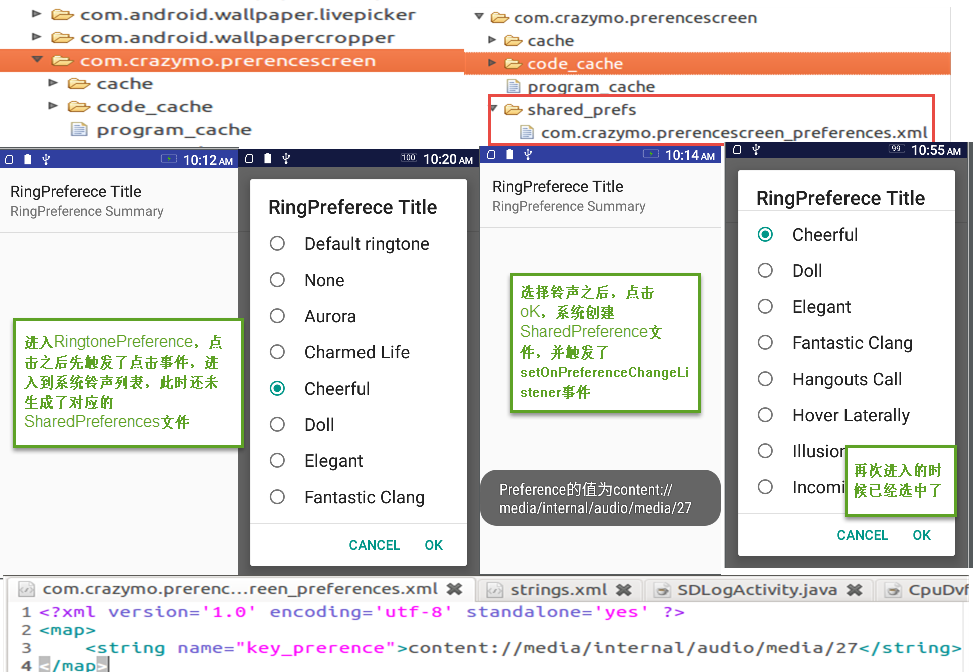
android:key="key\_prerence"

android:title="RingPreferece Title"

android:summary="RingPreference Summary"

/>

</PreferenceScreen>



# Preference的管理

F

Preference的管理主要包含Preference的**创建**、**添加**、**移除**和**寻找**特定Preference，Preference并没有直接提供相关替换的方法。

## 1、创建Preference

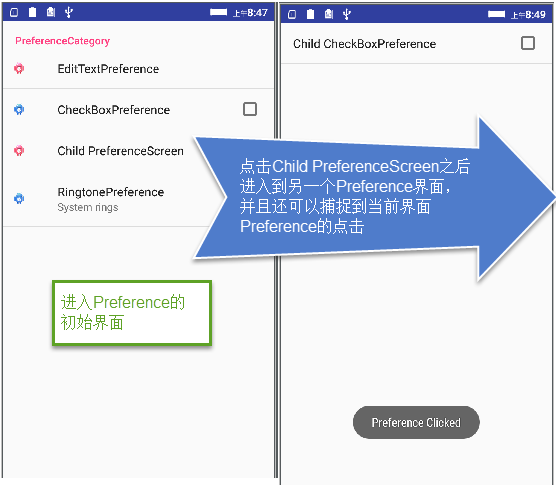
主要是通过各自对应Preference的构造方法或者直接在xml文件中定义来创建对应的Preference

## 2、addPreference添加Preference

添加只要是就是调用**PreferenceGroup的addPreference(Preference preference)**方法 来添加至容器PreferenceGroup。

## 3、findPreference寻找特定Preference

对于**PreferenceActivity**的**findPreference(key)**方法，我们只需要**知道key值就可以找到同一xml文件下相应的Preference，勿需考虑层级和嵌套关系**



## 4、removePreference或removeAll移除Preference

对于**PreferenceGroup**的**removePreference(Preference preference)**和**removeAll()**方法，都是针对某个PreferenceGroup来处理的，所以我们必须考虑层级嵌套关系,可以分为两步：**先找到PreferenceGroup和要删除的Preference**，再**调用PreferenceGroup的removePreference**执行删除动作。

首先这是我们的布局

首先这是我们的布局

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

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

android:key="key\_manage\_prefs">

<PreferenceCategory

android:key="key\_category"

android:title="PreferenceCategory">

<EditTextPreference

android:key="key\_edtprefs"

android:title="EditTextPreference" />

<PreferenceScreen

android:key="key\_child\_prefscreen"

android:title="Child PreferenceScreen">

<CheckBoxPreference

android:key="key\_checkbox"

android:title="CheckBoxPreference" />

</PreferenceScreen>

</PreferenceCategory>

</PreferenceScreen>

private void removePreferenceByKey(){

/\*PreferenceGroup preferenceGroup=(PreferenceGroup)findPreference("key\_category");//先找到PreferenceGroup

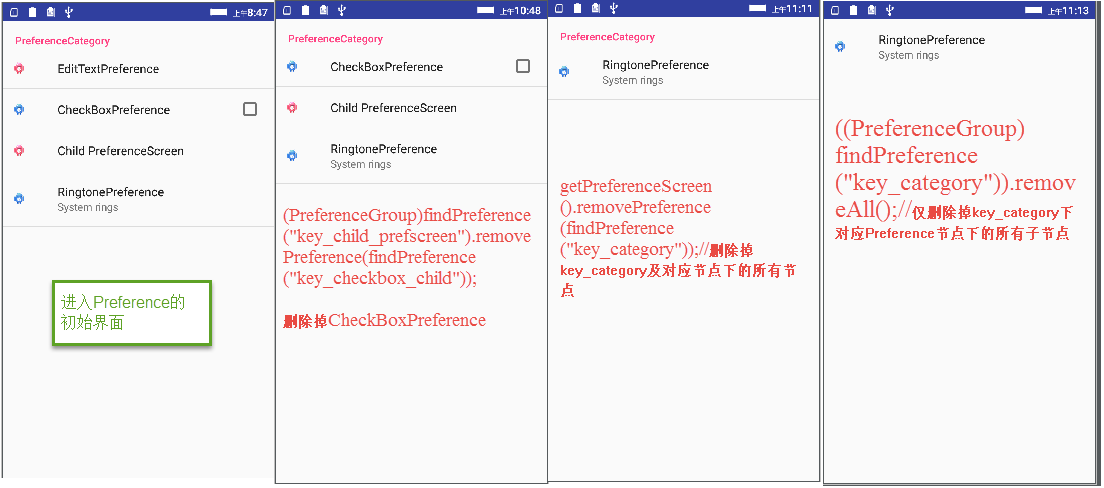
Preference preference=findPreference("key\_edtprefs");//再找到要删除的Preference\*/

/\*(PreferenceGroup)findPreference("key\_child\_prefscreen").removePreference(findPreference("key\_checkbox\_child"));//执行删除key为key\_checkbox\_child的Preference\*\*/

getPreferenceScreen().removePreference(findPreference("key\_category"));//删除掉key\_category及对应节点下的所有节点

//ERROR//getPreferenceScreen().removePreference(findPreference("key\_edtprefs"));//无效，因为getPreferenceScreen获得的是当前的顶级容器，而key\_edtprefs不是它的直接字节点

////((PreferenceGroup)findPreference("key\_category")).removeAll();//仅删除掉key\_category下对应Preference节点下的所有子节点



## 在XML中定义Preference

可以在运行时创建Preference也可以通过在xml中定义Preference,每一个Preference的子类都可以用XML中的元素来定义，在 res/xml/  
文件夹中创建一个文件如preference.xml，一般的布局如下：

## 创建PreferenceActivity

Sdf

###  在res/xml/文件夹中定义 preference.xml

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

<PreferenceScreen

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

<PreferenceCategory

android:title="@string/inline\_preferences">

<CheckBoxPreference

android:key="check\_box\_pref"

android:title="@string/title\_checkbox\_preference"

android:summary="@string/summary\_checkbox\_preference">

.....

</CheckBoxPreference>

</PreferenceCategory>

</PreferenceScreen>

###  Activity继承PreferenceActivity

public class SettingsActivity1 extends PreferenceActivity{

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

addPreferencesFromResource(R.xml.preference); //API 10之后的版本这个方法以及过时，推荐使用Fragment的方式

}

}

## 使用Preference Fragments

使用PreferenceFragment来显示Preference,就不需要继承PreferenceActivity  
继承PreferenceFragment在onCreate()方法中调用addPreferencesFromResource()方法

public class SettingsFragment extends PreferenceFragment{ @Override public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); this.addPreferencesFromResource(R.xml.preference); } }

创建一个Activity,然后将定义的Fragments添加到当前的Activity中,最终呈现的效果和PreferenceActivity是一样的

public class SettingsActivity2 extends Activity{ @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); this.getFragmentManager().beginTransaction() .replace(android.R.id.content, new SettingsFragment()) .commit(); } }

## 自定义Preference的布局

通过Preference的属性android:layout=""设置自定义的layout文件

## REF

[Android设置界面之Preference](https://www.jianshu.com/p/6d6f84e2f50d)

[android开发之PreferenceScreen使用详解](https://blog.csdn.net/u012702547/article/details/47039199)

## REF

# SwitchPreference

public SwitchPreference(Context context, AttributeSet attrs) {

this(context, attrs, com.android.internal.R.attr.switchPreferenceStyle);

}

switchPreferenceStyle

View checkableView = view.findViewById(com.android.internal.R.id.***switch\_widget***);

## Theme.xml

<**item name="switchPreferenceStyle"**>@style/Preference.DeviceDefault.SwitchPreference</**item**>

## Style\_ Material

<**style name="Preference.Material.SwitchPreference"**>  
 <**item name="widgetLayout"**>@layout/preference\_widget\_switch</**item**>  
 <**item name="switchTextOn"**>@string/capital\_on</**item**>  
 <**item name="switchTextOff"**>@string/capital\_off</**item**>  
</**style**>

preference\_widget\_switch.xml

**android:id="@+android:id/switch\_widget"**

**Switch**

## Preference

**case** com.android.internal.R.styleable.***Preference\_widgetLayout***:  
 **mWidgetLayoutResId** = a.getResourceId(attr, **mWidgetLayoutResId**);  
 **break**;

## <style name="Rm500.Theme.Settings" parent="Theme.Settings">

<**item name="android:switchStyle"**>@style/Rm500.Widget.Switch</**item**>

<style name="Rm500.Widget.Switch" **parent="@android:style/Widget.Material.CompoundButton.Switch"**>  
 <**item name="android:thumb"**>@drawable/switch\_thumb\_material\_anim</**item**><**item name="android:track"**>@drawable/switch\_track</**item**>  
 <**item name="android:trackTint"**>@color/white</**item**>  
 </**style**>

总结：分析这个的时候，必须了解源码框架

继承关系，构造函数

Framework部分res框架

# DropDownPreference

**public** DropDownPreference(Context context, AttributeSet attrs) {  
 **this**(context, attrs, R.attr.***dropdownPreferenceStyle***);  
}

## preference-v14-25.2.0

<**style name="PreferenceThemeOverlay.v14.Material"**>

**dropdownPreferenceStyle**

<**style name="Preference.DropDown.Material"**>  
 <**item name="android:layout"**>@layout/preference\_dropdown\_material</**item**>  
</**style**>

### Layout/preference\_dropdown\_material.mxl

<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:minHeight="?android:attr/listPreferredItemHeightSmall"  
 android:gravity="center\_vertical"  
 android:paddingLeft="?android:attr/listPreferredItemPaddingLeft"  
 android:paddingRight="?android:attr/listPreferredItemPaddingRight"  
 android:background="?android:attr/selectableItemBackground"  
 android:clipToPadding="false"  
 android:focusable="true"** >  
  
 <**Spinner  
 android:id="@+id/spinner"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:visibility="invisible"** />  
  
 <**LinearLayout  
 android:id="@+id/icon\_frame"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginLeft="-4dp"  
 android:minWidth="60dp"  
 android:gravity="start|center\_vertical"  
 android:orientation="horizontal"  
 android:paddingRight="12dp"  
 android:paddingTop="4dp"  
 android:paddingBottom="4dp"**>  
 <**android.support.v7.internal.widget.PreferenceImageView  
 android:id="@android:id/icon"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 app:maxWidth="48dp"  
 app:maxHeight="48dp"** />  
 </**LinearLayout**>  
  
 <**RelativeLayout  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="1"  
 android:paddingTop="16dp"  
 android:paddingBottom="16dp"**>  
  
 <**TextView android:id="@android:id/title"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:singleLine="true"  
 android:textAppearance="@style/Preference\_TextAppearanceMaterialSubhead"  
 android:ellipsize="marquee"** />  
  
 <**TextView android:id="@android:id/summary"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@android:id/title"  
 android:layout\_alignLeft="@android:id/title"  
 android:textAppearance="?android:attr/textAppearanceSmall"  
 android:textColor="?android:attr/textColorSecondary"  
 android:maxLines="10"** />  
  
 </**RelativeLayout**>  
  
 *<!-- Preference should place its actual preference widget here. -->* <**LinearLayout android:id="@android:id/widget\_frame"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:gravity="end|center\_vertical"  
 android:paddingLeft="16dp"  
 android:orientation="vertical"** />

**Spinner**

# 源码分析

## ****Preference源码浅析****

他不是一个View，但是组合管理了一个View和PreferenceManager

//可以看见，他不是一个View，但是组合管理了一个View和PreferenceManager

public class Preference implements Comparable<Preference> {

......

//各种属性

......

private PreferenceManager mPreferenceManager;

//重点关注，和自定义及Preference显示原理息息相关，preference就是下面列出的xml资源

private int mLayoutResId = com.android.internal.R.layout.preference;

private int mWidgetLayoutResId;

......

//各种getXXX及setXXX方法

......

### getView

/\*\*

\* Gets the View that will be shown in the {@link PreferenceActivity}.

\* 获取Preference的item显示view

\*/

/\*\*

\* Gets the View that will be shown in the {@link PreferenceActivity}.

\* 获取Preference的item显示view

\*/

public View getView(View convertView, ViewGroup parent) {

if (convertView == null) {

convertView = onCreateView(parent);

}

onBindView(convertView);

return convertView;

}

protected View onCreateView(ViewGroup parent) {

final LayoutInflater layoutInflater = (LayoutInflater) mContext.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

final View layout = layoutInflater.inflate(mLayoutResId, parent, false);

final ViewGroup widgetFrame = (ViewGroup) layout.findViewById(com.android.internal.R.id.widget\_frame);

if (widgetFrame != null) {

//mWidgetLayoutResId有专门的set方法可以设置或者重写

if (mWidgetLayoutResId != 0) {

//android:id/widget\_frame为mWidgetLayoutResId所对应的布局预留空间插入

layoutInflater.inflate(mWidgetLayoutResId, widgetFrame);

} else {

//默认实现是null的

widgetFrame.setVisibility(View.GONE);

}

}

return layout;

}

/\*\*

\* Binds the created View to the data for this Preference.

\* <p>

\* This is a good place to grab references to custom Views in the layout and

\* set properties on them.

\* <p>

\*/

protected void onBindView(View view) {

//设置子View相关属性

final TextView titleView = (TextView) view.findViewById(com.android.internal.R.id.title);

if (titleView != null) {

final CharSequence title = getTitle();

if (!TextUtils.isEmpty(title)) {

titleView.setText(title);

titleView.setVisibility(View.VISIBLE);

} else {

titleView.setVisibility(View.GONE);

}

}

......

//类似的各种子View设置操作，不再列出

}

可以看见，这个getView其实就是上面PreferenceFragment分析中Adapter中getView调运的Preference的getView。怎么样，串起来吧。也就是说Preference不是View，但是他提供View给ListView的每一个Item显示，其提供的View的基类布局（上面Preference类中mLayoutResId属性的值）如下：

### com.android.internal.R.layout.*preference*

asdf

**[java]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. <!-- Layout **for** a Preference in a PreferenceActivity. The
2. Preference is able to place a specific widget **for** its particular
3. type in the "widget\_frame" layout. -->
4. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
5. android:layout\_width="match\_parent"
6. android:layout\_height="wrap\_content"
7. android:minHeight="?android:attr/listPreferredItemHeight"
8. android:gravity="center\_vertical"
9. android:paddingEnd="?android:attr/scrollbarSize"
10. android:background="?android:attr/selectableItemBackground" >
12. <ImageView
13. android:id="@+android:id/icon"
14. android:layout\_width="wrap\_content"
15. android:layout\_height="wrap\_content"
16. android:layout\_gravity="center"
17. />
19. <RelativeLayout
20. android:layout\_width="wrap\_content"
21. android:layout\_height="wrap\_content"
22. android:layout\_marginStart="15dip"
23. android:layout\_marginEnd="6dip"
24. android:layout\_marginTop="6dip"
25. android:layout\_marginBottom="6dip"
26. android:layout\_weight="1">
28. <TextView android:id="@+android:id/title"
29. android:layout\_width="wrap\_content"
30. android:layout\_height="wrap\_content"
31. android:singleLine="true"
32. android:textAppearance="?android:attr/textAppearanceLarge"
33. android:ellipsize="marquee"
34. android:fadingEdge="horizontal" />
36. <TextView android:id="@+android:id/summary"
37. android:layout\_width="wrap\_content"
38. android:layout\_height="wrap\_content"
39. android:layout\_below="@android:id/title"
40. android:layout\_alignStart="@android:id/title"
41. android:textAppearance="?android:attr/textAppearanceSmall"
42. android:textColor="?android:attr/textColorSecondary"
43. android:maxLines="4" />
45. </RelativeLayout>
47. <!-- Preference should place its actual preference widget here. -->
48. <LinearLayout android:id="@+android:id/widget\_frame"
49. android:layout\_width="wrap\_content"
50. android:layout\_height="match\_parent"
51. android:gravity="center\_vertical"
52. android:orientation="vertical" />
54. </LinearLayout>

### widget\_frame

通过源代码和上边的注释也能了解到@+android:id/widget\_frame其实为mWidgetLayoutResId所对应的布局预留了空间。

**protected** [View](http://www.google.com/search?hl=en&q=allinurl%3Aview+java.sun.com&btnI=I%27m%20Feeling%20Lucky) onCreateView(ViewGroup parent) {

**final** LayoutInflater layoutInflater =

(LayoutInflater) mContext.getSystemService([Context](http://www.google.com/search?hl=en&q=allinurl%3Acontext+java.sun.com&btnI=I%27m%20Feeling%20Lucky).LAYOUT\_INFLATER\_SERVICE);

**final** [View](http://www.google.com/search?hl=en&q=allinurl%3Aview+java.sun.com&btnI=I%27m%20Feeling%20Lucky) layout = layoutInflater.inflate(mLayoutResId, parent, **false**);

**final** ViewGroup widgetFrame = (ViewGroup) layout

.findViewById(com.android.internal.R.id.widget\_frame);

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

**if** (mWidgetLayoutResId != 0) {

layoutInflater.inflate(mWidgetLayoutResId, widgetFrame);

} **else** {

widgetFrame.setVisibility([View](http://www.google.com/search?hl=en&q=allinurl%3Aview+java.sun.com&btnI=I%27m%20Feeling%20Lucky).GONE);

}

}

**return** layout;

}

### 小结

可以看出来，上面我们分析Preference的onCreateView、getView、onBindView这几个方法其实是整个Preference组件显示等的核心方法，所以正如系统提供的Preference的各种实用子类一样，当我们想自定义Preference的时候完全可以重写这些方法来得到自己的各种自定义View，这样就完美的解决了代码的扩展性，我们不用去修改ListAdapter的实现就能实现自定义的Preference，所以说可见Google的工程师在设计Preference结构时是多么的牛叉，不得不膜拜

# REF

[Android进阶——Preference详解之初识Preference及Preference系（一）](https://blog.csdn.net/crazymo_/article/details/51705942)

[Android进阶——Preference详解之Preference系的基本应用和管理(二)](https://blog.csdn.net/crazymo_/article/details/51496516)

[Android进阶——Preference详解之Preference系的基本应用(三)](https://blog.csdn.net/zhangyongfeiyong/article/details/79157552)