android如何在xml中引用内部类的View

2018年06月11日 16:51:31 weixin\_38503885 阅读数：139

https://blog.csdn.net/gorgle/article/details/51428515

上周，有个同事在xml中引用内部类的View时候出错，问我在xml中能不能用内部类的View，我以前项目曾经这样做过，因此当时很肯定地告诉他可以。看了下他的代码，xml中的class属性引用的内部类写法错了，把“$”写成“.”,我让他改下就可以。他试完之后告诉我还是不行，我瞬间懵逼了。当时因时间关系，没时间去查错，让他先改为外部类处理。今天早上有空查看下系统源码，终于把这个问题搞清楚了。进入今天的正题：

xml布局引用内部类View的正确写法

系统是如何根据class来创建View

xml布局引用内部类View的正确写法

解决问题从源码入手。首先从Activity的setContentView开始

public void setContentView(@LayoutRes int layoutResID) {

getWindow().setContentView(layoutResID);

initWindowDecorActionBar();

}

1

2

3

4

调用PhoneWindow的setContentView：

@Override

public void setContentView(int layoutResID) {

...

if (hasFeature(FEATURE\_CONTENT\_TRANSITIONS)) {

final Scene newScene = Scene.getSceneForLayout(mContentParent, layoutResID,

getContext());

transitionTo(newScene);

} else {

mLayoutInflater.inflate(layoutResID, mContentParent);

}

...

}

1

2

3

4

5

6

7

8

9

10

11

12

13

调用LayoutInflater的inflate方法调用顺序如下（已删除大部分无关代码）：

public View inflate(@LayoutRes int resource, @Nullable ViewGroup root, boolean attachToRoot) {

...

final XmlResourceParser parser = res.getLayout(resource);

try {

return inflate(parser, root, attachToRoot);

} finally {

parser.close();

}

}

public View inflate(XmlPullParser parser, @Nullable ViewGroup root, boolean attachToRoot) {

...

final View temp = createViewFromTag(root, name, inflaterContext, attrs);

...

}

View createViewFromTag(View parent, String name, Context context, AttributeSet attrs,

boolean ignoreThemeAttr) {

if (name.equals("view")) {

name = attrs.getAttributeValue(null, "class");

}

...

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

当看到createViewFromTag方法的name.equals(“view”)时候，我瞬间明白了，原来我同事把xml中tag写成大写View了，于是赶脚写个Demo测试一下：

内部类MyView：

package com.baidusoso.innerclassview;

...

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public static class MyView extends TextView {

public MyView(Context context, AttributeSet attrs) {

this(context, attrs,R.attr.CustomizeStyle);

Log.d(TAG, "MyView");

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

布局文件activity\_main.xml

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@android:color/transparent"

android:orientation="vertical" >

<view

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

class="com.baidusoso.innerclassview.MainActivity$MyView"

android:text="Hello world!!!" />

</LinearLayout>

1

2

3

4

5

6

7

8

9

10

11

12

Duang，程序果然运行起来了。

现在总结一下xml布局引用内部类View的正确写法：

1. xml布局文件中tag的view必须是：小写、小写、小写，重要的事情说3遍；

2. 内部类的View必须是静态的，因为普通内部类必须通过对象来引用，这在xml中是不可能的（如果看不明白这点，赶紧去学习下java内部类相关知识）

3. 引用类属性直接是class，没有如android:这样的名字空间；外部类和静态内部类是用$（而不是“.”）连起来的，如：

class=”com.baidusoso.innerclassview.MainActivity$MyView”

4. 静态内部类必须有带Context、AttributeSet这2个参数的构造函数，如

public MyView(Context context, AttributeSet attrs)

1

我将在下一节对第四点做出解释。

系统是如何根据class来创建View

那么写好class之后，系统是如何进行校验这个class是否存在？怎么构建其View对象呢？

带着这2个问题，我们接着往下看源码，还是在LayoutInflater的createViewFromTag

View createViewFromTag(View parent, String name, Context context, AttributeSet attrs,

boolean ignoreThemeAttr) {

if (name.equals("view")) {

name = attrs.getAttributeValue(null, "class");

}

...

if (view == null) {

final Object lastContext = mConstructorArgs[0];

mConstructorArgs[0] = context;

try {

if (-1 == name.indexOf('.')) {

view = onCreateView(parent, name, attrs);

} else {

view = createView(name, null, attrs);

}

} finally {

mConstructorArgs[0] = lastContext;

}

}

return view;

...

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

这里会根据class是否包含”.”调用2个不同的函数：onCreateView和createView

我们先来看onCreateView

protected View onCreateView(View parent, String name, AttributeSet attrs)

throws ClassNotFoundException {

return onCreateView(name, attrs);

}

protected View onCreateView(String name, AttributeSet attrs)

throws ClassNotFoundException {

return createView(name, "android.view.", attrs);

}

1

2

3

4

5

6

7

8

9

从代码中，我们得知onCreateView最后也是调用到createView，只是第二个参数是”android.view.”，而不是null。也就是说，如果class的值没带”.”，那默认就会到android.view这个包名下去找相应的类，如：

<view

android:layout\_width="match\_parent"

android:layout\_height="1dp"

class="View"

android:background="#000000" />

1

2

3

4

5

以上代码就是构建一个android.view.View对象，内容就是一根长度充满父节点的黑线。

接着我们再看看createView方法：

public final View createView(String name, String prefix, AttributeSet attrs)

throws ClassNotFoundException, InflateException {

Constructor<? extends View> constructor = sConstructorMap.get(name);

Class<? extends View> clazz = null;

try {

Trace.traceBegin(Trace.TRACE\_TAG\_VIEW, name);

if (constructor == null) {

// Class not found in the cache, see if it's real, and try to add it

clazz = mContext.getClassLoader().loadClass(

prefix != null ? (prefix + name) : name).asSubclass(View.class);

...

constructor = clazz.getConstructor(mConstructorSignature);

constructor.setAccessible(true);

sConstructorMap.put(name, constructor);

} else {

....

}

Object[] args = mConstructorArgs;

args[1] = attrs;

final View view = constructor.newInstance(args);

...

return view;

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

其中mContext.getClassLoader().loadClass方法就是加载class属性值，得到相应的类实例。如果我们把class值写错了，这里就会报ClassNotFoundException.这里我们解决本节开头提到的第一个问题：系统是如何进行校验这个class是否存在

创建class的类实例后，通过反射clazz.getConstructor(找到构造函数，其参数mConstructorSignature对应的值是：

static final Class<?>[] mConstructorSignature = new Class[] {

Context.class, AttributeSet.class};

1

2

现在，你明白了上节结论第四点提到对构造函数的要求：静态内部类必须有带Context、AttributeSet这2个参数的构造函数

如果我们定义的view中没有这个构造函数，那么就会抛出NoSuchMethodException。

接着通过final View view = constructor.newInstance(args);创建了View的实例。这也回答本节开头提到的第二个问题：怎么构建其View对象

最后再说一点，我们平常写的布局：

<LinearLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

1

2

3

也可以写出这样：

<view

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

class="android.widget.LinearLayout"

1

2

3

4

只是第一种写法显得很简单，简单就是美！