DataBinding的使用

皮卡搜 关注
2018.01.31 14:19:43 字数 2,236 阅读 3,274

这段有时间就看了下Databinding，简单记录一下Databinding的使用方式！

一、Databinding简单尝试

首先在Module app下build.gradle中配置databinding

```
android{
    ...

    dataBinding {
            enabled true
    }

    ...
}
```

easy！这样就配好了。

但是，重点还是在使用，最大的不同是布局文件，之前的布局文件是以LinearLayout、RelativeLayout...几大布局为根标签，使用Databinding的布局文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>

        <variable
            name="test"
            type="com.test.qby.newtestapplication.model.TestModel" />

    </data>

    <RelativeLayout style="@style/MatchMatch">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:text="@{@string/test+test.name}"
            android:textColor="@color/black"
            android:textSize="@dimen/sp_16" />

    </RelativeLayout>
</layout>
```

TestModel类：

```
package com.test.qby.newtestapplication.model;

/**
 * Created by qby on 2018/1/29 0029.
 * 测试
 */
public class TestModel {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

记得必须要有getter/setter方法，虽然必须要有，但不一定必须我们写哦，后面会提到。
MainTestActivity页面：

```
package com.test.qby.newtestapplication.ui;

import android.app.Activity;
import android.databinding.DataBindingUtil;
import android.os.Bundle;
import android.support.annotation.Nullable;

import com.test.qby.newtestapplication.R;
import com.test.qby.newtestapplication.databinding.ActivityMainTestBinding;
import com.test.qby.newtestapplication.model.TestModel;

/**
 * Created by qby on 2018/1/29 0029.
 * 测试页面
 */

public class MainTestActivity extends Activity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityMainTestBinding binding = DataBindingUtil.setContentView(this, R.layou
        TestModel testModel = new TestModel();
        testModel.setName("这是测试文字");
        binding.setTest(testModel);
    }
}
```

这样就简单的使用了databinding，注意自动生成的ActivityMainTestBinding命名是以对应布局的名字(activity_main_test)去除连接单词的下划线后(activitymaintest)，单词首字母大写(ActivityMainTest)+Binding，使用DataBindingUtil获取布局转为ActivityMainTestBinding 的时候，需要先Make module 'app'一下，先编译生成ActivityMainTestBinding，不然只能获取到ViewDataBinding，当然布局有错误的时候也是转不了的。

二、布局文件结构

上面布局文件的结构要写对<layout></layout>为根节点，里面包含一个<data></data>节点和以前布局写法的布局组件。
data节点下有<variable/>节点,variable有name、type属性，name为自己定义的type类型的对象，用来在布局文件中引用；type就是你要用到的类啦，比如上面写的JavaBean类或者点击事件的android.view.View.OnClickListener接口类。data节点下还有一个<import/>节点，写法如下：

```
<data>

    <import type="com.test.qby.newtestapplication.model.TestModel"/>

    <variable
        name="test"
        type="TestModel" />

</data>
```

此时variable属性type的值为import属性type的类名TestModel，这就可能会出现如果项目中在不同包名下有相同类名的类需要同时引用怎么办？google也给了解决办法：import还有一个alias属性，给导入的类起别名，比如：

```
<data>

    <import type="com.test.qby.newtestapplication.model.TestModel" alias="ModelTest"/>
    <import type="com.test.qby.newtestapplication.viewmodel.TestModel"/>

</data>
```

不想设置variable的话，引用的时候直接用@{ModelTest.getName()}和@{TestModel.getName()}，但是需要getName()为static方法，这样就可以解决类名冲突了。
布局文件中对data数据的引用：

```
android:text="@{@string/test+test.name}"
```

内容是以@{}包裹的，需要注意下android:text需要拼接字符串的话，字符串要写在values/strings.xml中，不然可能你写个取消、确定之类的没问题，长点的字符串一运行就GG了，具体原因不知道，可能是google希望规范点的写法，也可能是个bug，记得就好。
列一下几种常用数据的引用方法：

```
<data>
    <variable
        name="array"
        type="String[]" />
    <variable
        name="list"
        type="java.util.ArrayList&lt;String&gt;" />
    <variable
        name="map"
        type="java.util.HashMap&lt;String,String&gt;" />
    <variable
        name="arrayIndex"
        type="int" />
    <variable
        name="listIndex"
        type="int" />
    <variable
        name="key"
        type="String" />
</data>
<LinearLayout
    android:orientation="vertical"
    style="@style/MathchMathch">
    <TextView
        style="@style/MatchWrap"
        android:text="@{array[arrayIndex]}"/>
    <TextView
        style="@style/MatchWrap"
        android:text="@{list[listIndex]}"/>
    <TextView
        style="@style/MatchWrap"
        android:text="@{map[key]}"/>
</LinearLayout>
```

在页面中进行初始化设置

```
//假数据，理解就好
String[] arrays = new String[]{"数组1", "数组2", "数组3"};
binding.setArrayIndex(0);
binding.setArray(arrays);
ArrayList<String> list = new ArrayList<>();
list.add("列表1");
list.add("列表2");
list.add("列表3");
binding.setListIndex(1);
binding.setList(list);
HashMap<String, String> map = new HashMap<>();
map.put("hash1", "键值对1");
map.put("hash2", "键值对2");
map.put("hash3", "键值对3");
binding.setKey("hash3");
binding.setMap(map);
```

此外，@{}包裹内容还支持运算符写法：

```
<data>

    <variable
            name="str"
            type="String"/>
    <variable
```

```
            name="error"
            type="boolean"/>

</data>

<LinearLayout
    style="@style/MatchMatch"
    android:orientation="vertical">

    <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text='@{error ?"error": "没错"}'
            android:textColor="@color/black"
            android:textSize="@dimen/sp_16" />
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text='@{str ?? str}'
            android:textColor="@color/black"
            android:textSize="@dimen/sp_16" />

</LinearLayout >
```

在页面中设置

```
binding.setError(true);
binding.setStr(null);
```

运行程序得到的结果是只显示“error”。
三元运算符没什么说的，跟代码中用法一样，还有很多其他运算符，请自行发掘；
“A??B”表示对前面字符串A是否为空的判断，如果为空，不显示，否则显示后面的字符串B，这里空判断包括null和字符串trim()之后为""的判断。（就是说纯空格的字符串也是会作为空处理的）
布局控件添加id，当然也是可以使用的，而且不用使用butterknife，还更简单，性能更好。
布局文件跟平时使用的一样：

```
 android:id="@+id/tv"
```

代码中：

```
TextView tv = binding.tv;
```

这就获取到了TextView ，注解什么的都不需要，炒鸡煎蛋！
布局中的引用注意单双引号的组合搭配。

三、事件绑定

使用Databinding还可以在布局中设置用户交互事件。

按照以前的用法在布局文件中设置当然还是可以的，但是执行方法就只能写在activity了；
在这里分享一个小知识，我也是刚发现。

```
如果你要在TextView上设置点击事件，
最好不要使用
在TextView布局上  添加onClick="testClick"，
在Activity中  public void testClick(View view){}
这种方式，
因为这样在API19以下是不生效的。
如果想要使用这种方式的话，
可以把TextView换成Button，
或者在TextView的父控件LinearLayout、RelativeLayout...上配置onClick属性，
或者就对低版本系统不做兼容。
```

下面是Databinding绑定事件的用法：

布局中引用方式

1、跟之前的方式几乎没区别

```
<variable
    name="testClick"
    type="android.view.View.OnClickListener" />
...
//在控件中添加点击事件
android:onClick="@{testClick}"
...
```

这种引用方式，代码调用也只能用

```
binding.setTestClick(view -> Log.e(TAG, "点击测试"));
```

2、导入处理类、引用处理类的方法

```
<variable
    name="mHandler"
    type="com.test.qby.newtestapplication.listener.MyHandler" />
...
android:onClick="@{view->mHandler.testClick(view)}"
...
```

3、和2算是一种，variable一样，表达式用法不一样

```
android:onClick="@{mHandler::testClick}"
```

事件的调用及处理方式

1、新建事件的处理类

```
/**
 * Created by qby on 2018/1/30 0030.
 * 事件处理类
 */

public class MyHandler {
    private static final String TAG = "MyHandler";

    public void testClick(View view){
        Log.e(TAG,"点击测试");
    }
}
```

在页面调用

```
binding.setMHandler(new MyHandler());
```

这样写，适用于多个页面点击事件执行代码一样的情况，因为执行代码是写在MyHandler的testClick中的。

如果想执行不同代码，可以重写方法

或者（如下）

2、将MyHandler改为接口

```
/**
 * Created by qby on 2018/1/30 0030.
 * 事件处理接口
 */

public interface MyHandler {


```

```
    void testClick(View view);
}
```

在页面调用

```
binding.setMHandler(view -> Log.e(TAG, "点击测试"));
```

上面的引用方式2、3可以和处理方式1、2任意搭配

但是，这些只是控件自有属性的绑定，使用Databinding还可以简单的自定义属性。

四、自定义属性

Databinding提供了@BindingAdapter("属性名")注解来完成自定义属性。
在JavaBean中定义如下方法：

```
@BindingAdapter("show")
public static void showIcon(ImageView iv, String imgUrl) {
    if (!TextUtils.isEmpty(imgUrl)) {
        Glide.with(iv)
                .load(imgUrl)
                .into(iv);
    }
}
```

布局中引用时记得先加上自定义命名空间

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

在控件中使用

```
<ImageView
     style="@style/WrapWrap"
     app:show="@{test.url}"/>
```

这样，运行时就会自动加载TestModel中设置的url获取图片展示到控件上。

五、代码调用

页面中调用方式也有几种

```
//1
ActivityMainTestBinding binding = DataBindingUtil.setContentView(this, R.layout.activi
//2
View inflate = LayoutInflater.from(this).inflate(R.layout.activity_main_test, null);
ActivityMainTestBinding bind = DataBindingUtil.bind(inflate);
//3
ActivityMainTestBinding inflate = DataBindingUtil.inflate(LayoutInflater.from(this), R
```

根据情况自己选择方式就行，说了半天都是在扯，真正重要的在下面。

六、数据刷新

Databinding提供了三种通知方式来通过JavaBean更新UI，分别是Observable 对象，ObservableFilelds 字段和 Observable CIlections 集合,当这些数据对象绑定到 UI 以及数据对象的属性改变时，UI将自动更新。

第一种：JavaBean继承BaseObservable

```
public class TestModel extends BaseObservable{
    private String name;

    @Bindable
    public String getName() {
```

```
        return name;
    }

    public void setName(String name) {
        this.name = name;
        notifyPropertyChanged(BR.name);
    }

}
```

JavaBean继承BaseObservable；getter方法添加@Bindable注解；settter方法内部添加通知notifyPropertyChanged；

第二种：JavaBean中字段改为ObservableField
Databinding提供的类包括：ObservableField，ObservableBoolean，ObservableByte，ObservableChar，ObservableShort，ObservableInt，ObservableLong，ObservableFloat，ObservableDouble，ObservableParcelable。

```
public class TestModel {
    public ObservableField<String> name = new ObservableField<>();
    public ObservableBoolean name = new ObservableBoolean();
}
```

别忘了字段修饰符要public，调用直接使用set()

```
TestModel testModel = new TestModel();
testModel.name.set("这是测试文字");
testModel.isTrue.set(true);
```

布局中引用方式与其他的没区别，在代码中获取值就直接调用get()

```
String name = testModel.name.get();
boolean isTrue = testModel.isTrue.get();
```

这种方法虽然看着没有getter/setter方法，但是它是在内部实现过了。

第三种：不使用JavaBean，创建Observable Clollections
这种方式是为了不创建Javabean,而使用动态数据结构来更新UI。Databinding提供了ObservableArrayMap,ObservableArrayList两个类来实现。
使用代码动态创建

```
ObservableArrayList<Object> observableList = new ObservableArrayList<>();
observableList.add("databindingList");
observableList.add(12);
binding.setListIndex(1);
binding.setObservableList(observableList);

ObservableArrayMap<String,Object> observableMap = new ObservableArrayMap<>();
observableMap.put("hash1", "databindingMap");
observableMap.put("hash2", 16);
observableMap.put("hash3", "显示");
binding.setKey("hash3");
binding.setObservableMap(observableMap);
```

使用方式跟使用ArrayList、HashMap没什么区别

```
<data>

    <variable
        name="list"
        type="android.databinding.ObservableArrayList&lt;Object&gt;" />
    <variable
        name="map"
        type="android.databinding.ObservableArrayMap&lt;String,Object&gt;" />
    <variable
        name="listIndex"
        type="int" />
```

```
    <variable
        name="key"
        type="String" />
</data>
<LinearLayout
    android:orientation="vertical"
    style="@style/MathchMathch">
    <TextView
        style="@style/MatchWrap"
        android:text="@{String.valueOf(list[listIndex])}"/>
    <TextView
        style="@style/MatchWrap"
        android:text="@{String.valueOf(map[key])}"/>
</LinearLayout>
```

七、数据转换

如六所示，在布局中引用的数据，如果要求的类型与获取的类型不一致，需要开发者自行转换。
android:text="要求是CharSequence类型"，使用String.valueOf()转换；
还有一种非常实用的转换：

```
<TextView
     style="@style/WrapWrap"
     android:text="@{String.valueOf(observableMap[key])}"
     android:visibility="@{error?android.view.View.VISIBLE:android.view.View.INVISIBLE
     android:textColor="@{error?@color/red:@color/green}"
     android:background="@{error?@color/green:@color/red}"
     />
```

这里根据error是否为true，判断了TextView的显示隐藏、背景以及文字的颜色。
android:visibility中可以将android.view.View使用import导包，写成

```
//data节点下
<import type="android.view.View"/>
//引用
android:visibility="@{error?View.VISIBLE:View.INVISIBLE}"
```

注意同一属性只能引用同种类型的数据，即android:background中error为true/false，引用@color只能引用@color，引用@drawable只能引用@drawable，不能同时@color和@drawable混用（可能你混用还是能运行，但是一改变error的值切换的时候就会“嘣嘣嘣”了）。

自动转换
Databinding提供了自动转换注解@BindingConversion，还是来个栗子：

布局：

```
<variable
     name="time"
     type="java.util.Date"/>

<TextView
     style="@style/WrapWrap"
     android:text="@{time}"/>
```

代码：

```
binding.setTime(new Date());
```

直接运行，当然会报错，关键在这：

```
/**
 * Created by qby on 2018/1/31 0031.
 * 自动转换器
 */

```

```
public class MyDateConverter {

    @BindingConversion
    public static String convertDate(Date date) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd", Locale.CHINA);
        return sdf.format(date);
    }

}
```

convertDate方法你可以随意放，在哪个类中都行，主要是@BindingConversion注解会在代码编译的时候，布局中任意与convertDate参数类型一致的引用，先执行convertDate方法，再将返回值赋给引用。

八、列表适配器

ListView的适配器使用Databinding:

```
public class MyListViewAdapter<T> extends BaseAdapter {
    private LayoutInflater inflater;
    private int layoutId;
    private int variableId;
    private List<T> list;

    public FutureWeatherAdapter(Context context, int layoutId, List<T> list, int resId
        this.layoutId = layoutId;
        this.list = list;
        this.variableId = resId;
        inflater = LayoutInflater.from(context);
    }

    @Override

    public int getCount() {
        return list.size();
    }

    @Override
    public Object getItem(int position) {
        return list.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ViewDataBinding dataBinding;
        if (convertView == null) {
            dataBinding = DataBindingUtil.inflate(inflater, layoutId, parent, false);
        } else {
            dataBinding = DataBindingUtil.getBinding(convertView);
        }
        dataBinding.setVariable(variableId, list.get(position));
        return dataBinding.getRoot();
    }
}
```

dataBinding.getRoot()获取到的就是convertView；dataBinding.setVariable跟上面在页面写的binding.setTime一样，只不过setTime是因为已经转换成特定唯一的ActivityMainTestBinding，内部也是调用的setVariable，因为这个是公共ListView适配器，只能获取到抽象类ViewDataBinding，所以只能使用setVariable。variableId就相当于BR.time，这里与条目布局文件需要的对象一致。
例如 条目布局中：

```
<variable
     name="test"
     type="com.test.qby.newtestapplication.model.TestModel" />
```

这里就传BR.test，adapter使用的时候跟平时一样。

RecyclerView的适配器使用Databinding:

```
public class MyRecyclerViewAdapter<T> extends RecyclerView.Adapter {

    private LayoutInflater inflater;
    private int layoutId;
    private int variableId;
    private List<T> list;

    public MyRecyclerViewAdapter(Context context, int layoutId, List<T> list, int resI
        this.layoutId = layoutId;
        this.list = list;
        this.variableId = resId;
        inflater = LayoutInflater.from(context);
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        ViewDataBinding binding = DataBindingUtil.inflate(inflater, layoutId, viewGrou
        return new MyViewHolder(binding);
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
        if (holder instanceof MyViewHolder) {
            MyViewHolder holder1 = (MyViewHolder) holder;
            holder1.getBinding().setVariable(variableId, list.get(position));
            holder1.getBinding().executePendingBindings();
        }
    }

    @Override
    public int getItemCount() {
        return list.size();
    }

    public static class MyViewHolder extends RecyclerView.ViewHolder {

        private ViewDataBinding binding;

        public MyViewHolder(ViewDataBinding binding) {
            super(binding.getRoot());
            this.binding = binding;
        }

        public ViewDataBinding getBinding() {
            return this.binding;
        }
    }
}
```

Databinding在RecyclerView中的使用跟ListView中基本一样，出现了一个executePendingBindings方法是因为：

```
当数据改变时，binding会在下一帧去改变数据，如果我们需要立即改变，就去调用executePendingBindings方
```

在这里的作用就是让改变数据立即执行。

好了，本文到这就结束了，Databinding常用的这些就差不多了，没写到的地方用的时候去android官网查阅吧。