转载请注明链接: https://blog.csdn.net/feather_wch/article/details/88597266

DataBinding的基本使用

版本号:2019-03-23(13:20)

- DataBinding的基本使用
  - 集成DataBinding
    - SDK Manager中进行下载
    - build.gradle中开启
  - 布局中的使用和绑定表达式
    - 基础知识
      - import
      - 类型别名
      - 导入其它类
        - context
      - include
      - 基本使用
      - 报错: 找不到android.databinding
    - 数据绑定和注入
      - Fragment、RecyclerView中使用数据绑定
    - 表达式
      - DataBinding对于空指针异常的处理
      - 报错:Identifiers must have user defined types from the XML file. View is missing it
    - 集合的使用
    - 资源的使用

      * plural

      * 需要改写形式的资源
    - 事件处理
      - 方法引用
      - 绑定监听器
  - 处理可观察的对象
    - 可观察字段
    - 可观察集合
    - 可观察对象
  - 绑定类
    - 初始化
    - 控件的ID

        - ViewStubs
        - 动态变量
        - 自定义binding类的名称
    - Binding Adapters
        - 注解@BindingMethods
        - 注解@BindingAdapter
            - 多个参数: 利用Glide加载图片，在错误情况展示默认图片。
                - requireAll=false
        - 注解@BindingConversion
    - 数据的双向绑定
        - 自定义属性
        - Converters转换器
        - 系统内置的支持双向绑定的属性
    - 知识扩展
        - @plurals
        - gradlew进行报错定位
    - 问题汇总
    - 参考资料

集成DataBinding

SDK Manager中进行下载

1、SDK Manager的Tools中下载 `Android Support Repository`

build.gradle中开启

2、根目录build.gradle中增加能下载的仓库

```
maven { url 'http://maven.aliyun.com/nexus/content/groups/public/' }
maven { url 'http://maven.aliyun.com/nexus/content/repositories/jcenter' }
maven { url 'http://maven.aliyun.com/nexus/content/repositories/google' }
maven { url 'http://maven.aliyun.com/nexus/content/repositories/gradle-plugin' }
```

3、app的build.gradle中开启databinding

```
dataBinding{
    enabled = true
}
```

4、AS 3.1开始使用新的Databinding 增加编译器

> gradle.properties

```
android.databinding.enableV2=true
```

布局中的使用和绑定表达式

基础知识

import

1、import用于导入需要用的类，避免在xml布局中报错

```
<data>
    <import type="android.view.View"/>
</data>
```

> 就可以使用View了

```
<TextView
   android:text="@{user.lastName}"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:visibility="@{user.isAdult ? View.VISIBLE : View.GONE}"/>
```

类型别名

2、如果导入的两个类名称一致，避免混乱，应该使用别名

```
// View
<import type="android.view.View"/>
// 还是View
<import type="com.example.real.estate.View"
        alias="Vista"/>
```

> 用Vista就能代指 `com.example.real.estate.View`

导入其它类

3、导入其它类

```
<data>
    <import type="com.example.User"/>
    <import type="java.util.List"/>
    <variable name="user" type="User"/>
    <variable name="userList" type="List<User>"/>
</data>
```

4、也可以进行类型转换

```
<TextView
   android:text="@{((User)(object)).lastName}"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
```

context

5、布局中有个特殊的变量为context，是根View的getContext()获得的对象

> 该名 `context` 通过显式的variable，可以被重写掉。

include

1、databinding支持从父布局将变量传递到内部include包含的布局中

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:bind="http://schemas.android.com/apk/res-auto">
   <data>
       <variable name="user" type="com.example.User"/>
   </data>

<!-- WORK! -->
   <LinearLayout
       android:orientation="vertical">

       <include layout="@layout/name"
           bind:user="@{user}"/>
       <include layout="@layout/contact"
           bind:user="@{user}"/>

   </LinearLayout>
</layout>
```

> 但是不支持 `merge`标签

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:bind="http://schemas.android.com/apk/res-auto">
   <data>
       <variable name="user" type="com.example.User"/>
   </data>

<!-- Doesn't work -->
   <merge>
       <include layout="@layout/name"
           bind:user="@{user}"/>
       <include layout="@layout/contact"
           bind:user="@{user}"/>
   </merge>

</layout>
```

基本使用

1、Databinding的使用是在布局中通过标签 `layout`、`data`、`variable` 实现

> 1-使用如下: 将用户类User，作为别名user。将账号和密码和对应控件绑定。

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable
            name="user"
            type="com.hao.architecture.User"/>
    </data>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <Button
            android:id="@+id/account_btn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@{user.account}"/>

        <Button
            android:id="@+id/password_btn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@{user.password}"/>

    </LinearLayout>
</layout>
```

> 2-User类

```
package com.hao.architecture;

public class User {
    String account;
    String password;

    public String getAccount() {
        return account;
    }

    public void setAccount(String account) {
        this.account = account;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

报错: 找不到android.databinding

1、报错的原因是，后来更改了整体的包名，导致生成的databiding类出错

> 将整个项目clean，重新生成文件

数据绑定和注入

1、因为布局是 `activity_databinding_layout.xml` 因此生成的类是 `ActivityDatabindingLayoutBinding`

2、Activity的onCreate中使用DataBinding

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //  下面的这句话已经废弃，不再使用
        //setContentView(R.layout.activity_databinding_layout);
        // 1、使用该方法
        ActivityDatabindingLayoutBinding binding = DataBindingUtil.setContentView(this, R.layou
        User user = new User("feather", "123456");
        // 2、绑定数据
        binding.setUser(user);
    }
```

Fragment、RecyclerView中使用数据绑定

3、Fragment、RecyclerView中使用数据绑定，可以使用下面的方法

```
ListItemBinding binding = ListItemBinding.inflate(layoutInflater, viewGroup, false);


ListItemBinding binding = DataBindingUtil.inflate(layoutInflater, R.layout.list_item, viewGroup
```

表达式

1、DataBinding在xml的布局中还可以使用表达式

2、使用字符串拼接

> 显示效果为: 账号:xxxxx

```
        <Button
            xxx
            android:text='@{@string/account+ ":" + user.account}'/>
```

> 1. String.xml需要有对应词条: `<string name="account">账号</string>`
> 2. 外层一定要是单引号，不然内层的 "文本" 会导致编译报错。

3、数学运算、位运算、位移、比较都可以使用

```
+ - / * %
&& ||
& | ^
+ - ! ~
>> >>> <<
== > < >= <= (Note that < needs to be escaped as &lt;)
```

4、instanceof的使用

5、控制控件可见还是不可见

> 1-设置visibility

```
<Button
    android:visibility="@{user.age >= 18 ? View.VISIBLE : View.GONE}"/>
```

> 2-引入View

```
<data>
    <variable
        name="user"
        type="com.hao.architecture.User"/>
    <import type="android.view.View" />
</data>
```

6、如果User的account为null的处理

> 1. 如果不做任何处理，会导致显示文本为 `null`
> 2. 使用三元操作符进行处理

```
android:text='@{user.account != null ? user.account : ""}'
```

> 3-使用聚合的null判断，和上面等效

```
android:text='@{user.account ?? ""}'
```

DataBinding对于空指针异常的处理

1、DataBinding会自动处理空指针异常

> 1. 如果String为null，就赋值为 `null`
> 2. 如果int为空，就赋值为 `0`

报错:Identifiers must have user defined types from the XML file. View is missing it

1、解决原因：在设置控件的显示状况时，使用了View，却没有引入

> 引入View

```
    <data>
        <variable
            name="user"
            type="com.hao.architecture.User"/>
        <import type="android.view.View" />
    </data>
```

集合的使用

1、Databinding中还可以使用集合，如: List、Map、SparseArray等等

> 1-布局中引入。 记住<这个符号需要使用<来表示

```
    <data>
        <import type="android.util.SparseArray"/>
        <import type="java.util.Map"/>
        <import type="java.util.List"/>

        <variable name="list" type="List&lt;String>"/>
        <variable name="sparse" type="SparseArray&lt;String>"/>
        <variable name="map" type="Map&lt;String, String>"/>
        <variable name="index" type="int"/>
        <variable name="key" type="String"/>
    </data>
```

> 2-布局中使用

```
android:text="@{list[index]}"
…
android:text="@{sparse[index]}"
…
android:text="@{map[key]}"
```

> 3-Activity中注入数据

```
// HashMap，其他的同理
        HashMap<String, String> hashMap = new HashMap<>();
        hashMap.put("password", "1234567890");

        binding.setMap(hashMap); // HashMap
        binding.setKey("password"); // Key值
```

资源的使用

plural

1、plural复数形式字符串的使用

```
// 复数形式字符串
  Have an orange
  Have %d oranges

// 使用
android:text="@{@plurals/orange(orangeCount, orangeCount)}"
```

需要改写形式的资源

1、需要改写形式的资源

Type	Normal reference	Expression reference

Type	Normal reference	Expression reference
String[]	@array	@stringArray
int[]	@array	@intArray
TypedArray	@array	@typedArray
Animator	@animator	@animator
StateListAnimator	@animator	@stateListAnimator
color int	@color	@color
ColorStateList	@color	@colorStateList

事件处理

1、View的事件处理对应的方法有哪些

Class	Listener setter	Attribute
View	setOnClickListener(View.OnClickListener)	android:onClick
SearchView	setOnSearchClickListener(View.OnClickListener)	android:onSearchClick
ZoomControls	setOnZoomInClickListener(View.OnClickListener)	android:onZoomIn
ZoomControls	setOnZoomOutClickListener(View.OnClickListener)	android:onZoomOut

方法引用

2、方法引用的好处

> 1. 可以在编译时间被检查，如果方法不存在，或者签名错误
> 2. 绑定的监听器是在数据被绑定之后才真正创建了监听器的内部实现，而不是事件触发的时候。
> 3. 如果你希望在事件发生时，使用当时动态绑定的新数据, 以及任意使用各种表达式，应该用 监听器绑定 的方式

3、方法引用的使用方法

> 1-布局中引入

```
<variable name="handlers" type="com.hao.architecture.DataBindingActivity.ClickHandlers"/>
```

> 2-布局中使用该监听器

```
<Button
      android:onClick="@{handlers::onClickPassword}"/>
```

> 3-代码中定义监听器

```
public class DataBindingActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // xxx
    }

    public class ClickHandlers{
        public void onClickPassword(View view){
            Toast.makeText(DataBindingActivity.this, "密码已经验证", Toast.LENGTH_SHORT).show()
        }
    }
}
```

> 4-设置监听器

```
binding.setHandlers(new ClickHandlers());
```

绑定监听器

3、绑定监听器和方法引用的区别在于，绑定监听器可以执行任意的表达语句

4、监听器绑定的基本用法

> 1-xml, theView代表当前的View控件

```
<variable name="handlers" type="com.hao.architecture.DataBindingActivity.ClickHandlers"/>

<Button
    android:onClick="@{(theView)->handlers.onClickPassword(theView)}"/>
```

> 2-Java

```
binding.setHandlers(new ClickHandlers());
```

5、使用当前绑定的数据的内容

> 1-监听器, 使用了用户的数据。

```
    public class ClickHandlers{
        public void onClickPassword(User user){
            Toast.makeText(DataBindingActivity.this, user.password + ": 密码已经验证", Toast.LEI
        }
    }
```

> 2-xml中进行指定

```
<Button
    android:onClick="@{()->handlers.onClickPassword(user)}"/>
```

> 3-Java中设置

```
binding.setHandlers(new ClickHandlers());
```

6、利用监听器绑定使用复杂的表达式

> 1. View可见，做一些事。
> 2. View不可见，什么都不做。

```
android:onClick="@{(v) -> v.isVisible() ? doSomething() : void}"
```

处理可观察的对象

1、Data Binding可以让我们的数据对象拥有能通知其他对象的能力。共有三类可观察的目标

> 1. 对象
> 2. 字段
> 3. 集合

可观察字段

1、可观察的字段有如下9种

> 1.ObservableBoolean-布尔值
> 1.ObservableByte
> 1.ObservableChar
> 1.ObservableShort
> 1.ObservableInt
> 1.ObservableLong
> 1.ObservableFloat

> 1.ObservableDouble
> 1.ObservableParcelable-可序列化

2、ObservableField解决String常量导致无法更新UI的问题

> 1-xml布局中

```
<variable name="password"
          type="android.databinding.ObservableField&lt;String&gt;"/>
<Button
          android:text='@{password}' />
```

> 2-Java中更改String的值

```
// 1、初始值
ObservableField<String> password = new ObservableField<>("old password");
// 2、绑定
binding.setPassword(password);
// 3、设置新值
password.set("new password");
```

3、可观察字段的实例：使用 `public final` 避免访问操作的装箱和拆箱

```
private static class User {
    public final ObservableField<String> firstName = new ObservableField<>();
    public final ObservableField<String> lastName = new ObservableField<>();
    public final ObservableInt age = new ObservableInt();
}
// 设置值
user.firstName.set("Google");
// 获取值
int age = user.age.get();
```

可观察集合

4、可观察集合: ObservableMap

> 1-使用ObservableMap展示账号和密码

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <import type="android.databinding.ObservableMap"/>
        <variable name="user" type="ObservableMap&lt;String, Object>"/>
    </data>
    <LinearLayout
        android:orientation="vertical">

        <Button
            android:text='@{user.account}'/>

        <Button
            android:text='@{String.valueOf(user.age)}' />

    </LinearLayout>
</layout>
```

> 2-Java中填充入数据

```
        ObservableArrayMap<String, Object> user = new ObservableArrayMap<>();
        user.put("account", "feather");
        user.put("password", "123456");
        user.put("age", 17);
        binding.setUser(user);
```

5、可观察的列表: ObservableArrayList

> 1-xml

```
    <data>
        <import type="android.databinding.ObservableArrayList"/>
        <variable name="user" type="ObservableArrayList&lt;Object>"/>
    </data>

    <Button
        android:text='@{user[0]}'/>
    <Button
        android:text='@{String.valueOf(user[1])}' />
```

> 2-ObservableArrayList展示数据

```
        ObservableArrayList<Object> user = new ObservableArrayList<>();
        user.add("feather");
        user.add(20);
        binding.setUser(user);
```

可观察对象

6、可观察对象是什么?

1. 一个类实现了 `Observable接口` ，允许进行监听器的注册，并且提供了通知的能力。
2. 为了方便使用，应该直接使用 `BaseObservable` 。内部完成了监听器注册和移除的逻辑。

7、BaseObservable的使用

1. 使用 `@Bindable` 注解 `getter`
2. 在 `setter` 中调用 `notifyPropertyChanged()`

```
private static class User extends BaseObservable {
    private String firstName;
    private String lastName;

    @Bindable
    public String getFirstName() {
        return this.firstName;
    }

    @Bindable
    public String getLastName() {
        return this.lastName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
        notifyPropertyChanged(BR.firstName);
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
        notifyPropertyChanged(BR.lastName);
    }
}
```

8、BR是什么?

1. BR是 `DataBinding`自动生成的类 ， 包含了所有 需要数据绑定的资源`ID`
2. `@Bindable` 注解在编译期间，会在 `BR类` 中产生一个Entry

9、PropertyChangeRegistry的作用

绑定类

1、绑定类是什么?

1. 在xml中配置好之后， `DataBinding` 会自动产生对应的类，如:
   `ActivityDatabindingLayoutBinding`
2. 该类继承自 `ViewDataBinding`

初始化

2、绑定类的初始化

```
// 1、第一种
View viewRoot = LayoutInflater.from(this).inflate(layoutId, parent, attachToParent);
ViewDataBinding binding = DataBindingUtil.bind(viewRoot);

// 2、第二种：用于Fragment，或者List的Adapter中
ListItemBinding binding = ListItemBinding.inflate(layoutInflater, viewGroup, false);
// or
ListItemBinding binding = DataBindingUtil.inflate(layoutInflater, R.layout.list_item, viewGroup

// 3、将inflate和绑定分离
MyLayoutBinding binding = MyLayoutBinding.bind(viewRoot);
```

控件的ID

3、DataBinding会为布局中具有ID的控件在产生的绑定类中创建一个不可变的字段

> 1. 例如下面的View，会产生名称和ID一样的字段。firstName和lastName
> 2. 这个机制比调用 `findViewById()` 更快

```
    <Button
        android:id="@+id/account"/>

    <Button
        android:id="@+id/password"/>
```

> 从Binding类中获取到同名的控件，比findViewById更快。

```
ActivityDatabindingLayoutBinding binding = DataBindingUtil.setContentView(this, R.layout.activi
// 名为account的Button
Button account = binding.account;
```

ViewStubs

1、DataBinding在生成的Binding类中使用 `ViewStubProxy` 代表 `ViewStub`

> 1. `ViewStubProxy` 必须监听 `ViewStub`的`OnInflateListener`， 因为ViewStub一开始并不是在View的层级中的
> 2. 可以在 `ViewStubProxy` 上设置一个 `OnInflateListener`，在建立绑定后调用。

动态变量

2、RecyclerView中需要用到动态变量

> 1. 需要在 调用onBindViewHolder() 方法后才能分配绑定的数值
> 2. 需要 重新设置变量 并且 立即进行绑定 (默认是在下一帧的时候绑定，但是这个时候需要立即绑定)

```
public void onBindViewHolder(BindingHolder holder, int position) {
    final T item = items.get(position);
    // 1、重新设置变量
    holder.getBinding().setVariable(BR.item, item);
    // 2、立即进行绑定
    holder.getBinding().executePendingBindings();
}
```

自定义binding类的名称

1、自定义绑定类的名称

```
// 1、自定义类名
<data class="ContactItem">
    …
</data>

// 2、更改类所在的包
<data class=".ContactItem">
    …
</data>
<data class="com.example.ContactItem">
    …
</data>
```

Binding Adapters

1、属性 android:text="@{user.name}" 内部是什么处理机制?

> 1. 首先找到了 setText(xxx) 这个方法，和 text= 相匹配
> 2. 再根据参数 user.name 匹配到对应的 user.getName() 这个方法

2、如果没有对应的属性，利用Databinding也可以为任何setter创造出属性

> 例如: DrawerLayout, 没有多余的属性

```
<android.support.v4.widget.DrawerLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:scrimColor="@{@color/scrim}"
    app:drawerListener="@{fragment.drawerListener}">
```

> 1. 会自动使用 `setScrimColor(int)` 和 `setDrawerListener(DrawerListener)` 方法
> 2. 即使没有对应的这些属性。

注解@BindingMethods

3、如果一些属性具有的setter和其属性名不匹配，就可以使用 `BindingMethods` 注解

> 1. `@BindingMethods` 注解中，还可以包含多个 `@BindingMethod`注解
> 2. 将ImageView的hint属性和 `setImageTintList` 相关联

```
@BindingMethods({
       @BindingMethod(type = "android.widget.ImageView",
                      attribute = "android:tint",
                      method = "setImageTintList"),
})
```

注解@BindingAdapter

4、有的时候原有的属性名需要绑定上自定义的逻辑

> 1-假如: 某些View没有 `android:paddingLeft` 对应的设置器，只有方法 `setPadding(left, top, right, bottom)`
> 2-通过下面方法可以进行自定义。第一个参数：决定该和该属性关联的View的类型。第二个参数：决定了该绑定表达式接收的参数类型。

```
@BindingAdapter("android:paddingLeft")
public static void setPaddingLeft(View view, int padding) {
  view.setPadding(padding,
                  view.getPaddingTop(),
                  view.getPaddingRight(),
                  view.getPaddingBottom());
}
```

多个参数: 利用Glide加载图片，在错误情况展示默认图片。

1、现在有一个需求，利用Databinding动态传入不同url，利用Glide加载图片后在绑定的ImageView上展示

> 1-xml

```
        <ImageView
            android:id="@+id/head_portrait"
            xxx
            app:imageUrl="@{user.headUrl}"
            app:imageError="@{@drawable/googlelogo}"/>
```

> 2-@BindingAdapter

```
// DataBindingActivity.java
    @BindingAdapter({"imageUrl", "imageError"})
    public static void loadImage(ImageView view, String url, Drawable error){
        Glide.with(view).load(url).error(error).into(view);
    }
```

> 3-在Activity中进行绑定

```
public class DataBindingActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ActivityDatabindingLayoutBinding binding = DataBindingUtil.setContentView(this, R.layou

        // 指定头像的URL
        User user = new User("account", "password", "https://github.com/bumptech/glide/raw/mast
        binding.setUser(user);
    }
}
```

> 4-使用该方法的注意点：**必须 `app:imageUrl` 和 `app:imageError`**

requireAll=false

3、如何在指定一个属性的时候就触发绑定的Adapter?

> 1. 使用 `@BindingAdapter` 的属性 `requireAll=false`

```
    @BindingAdapter(value={"imageUrl", "imageError"}, requireAll = false)
    public static void loadImage(ImageView view, String url, Drawable error){
        Glide.with(view).load(url).error(error).into(view);
    }
```

> 2-xml中只使用一个属性，也能匹配到

```
<ImageView
    android:id="@+id/head_portrait"
    app:imageUrl="@{user.headUrl}"/>
```

注解@BindingConversion

1、@BindingConversion的作用：自定义类型转换

> 1. 例如 `android:bakcground` 需要的是 `Drawable` ， 但是传入的是 `int color` 就需要进行类型转换
> 2-xml

```
<View
   android:background="@{isError ? @color/red : @color/white}"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
```

> 3-Java

```
@BindingConversion
public static ColorDrawable convertColorToDrawable(int color) {
    return new ColorDrawable(color);
}
```

2、DataBinding严禁使用不同的数据类型，如下:

```
android:background="@{isError ? @drawable/error : @color/white}"
```

数据的双向绑定

1、什么是数据的双向绑定?

> 1. 数据发生变化后，去通知UI进行改变。
> 2. UI改变后，反过去通知数据层，比如进行数据的保存等操作。

2、数据双向绑定的简单实例

> 1-使用形如"@={xxx}"进行双向绑定, 如下:

```
        <CheckBox
            android:id="@+id/remember_me_cb"
            xxx
            android:checked="@={user.remember}"/>
```

> 2-在Bean中实现 `BaseObservable` 接口，并且使用 `@Bindable`
> 1. set和get方法必须要匹配字段名，不然会报错。
> 2. 在set方法中进行特殊操作，然后执行 `notifyPropertyChanged` 再反过去改变UI的操作。

```
public class User extends BaseObservable {
    boolean remember;

    public User(boolean remember) {
        this.remember = remember;
    }

    @Bindable
    public boolean getRemember() {
        return remember;
    }

    public void setRemember(boolean remember) {
        if(this.remember != remember) {
            this.remember = remember;

            // 根据UI的改变，做出一些操作。例如存储账号和密码。这里是设置为未选中状态
            if(this.remember == true){
                this.remember = false;
                // 通知属性已经改变
                notifyPropertyChanged(BR.remember);
            }
        }
    }
}
```

自定义属性

1、自定义属性的双向绑定

> 1-setter，注意：必须要打破潜在的无限循环。--- 数据改变

```
@BindingAdapter("time")
public static void setTime(MyView view, Time newValue) {
    // Important to break potential infinite loops.
    if (view.time != newValue) {
        view.time = newValue;
    }
}
```

> 2-getter --- View的属性改变

```
@InverseBindingAdapter("time")
public static Time getTime(MyView view) {
    return view.getTime();
}
```

2、对自定义View使用自定义的监听器

```
@BindingAdapter("app:timeAttrChanged")
public static void setListeners(
        MyView view, final InverseBindingListener attrChange) {
    // Set a listener for click, focus, touch, etc.
}
```

> 双向绑定的DataBinding会自动生成一个属性，例如 `app:timeAttrChanged` ， 是原属性加上后缀拼接成的 `app:xxxAttrChanged`

Converters转换器

1、Converter转换器的使用

> 1-有可能输入的数据是long值，但是需要展示的是日期的字符串，就需要进行转换
> 2-转换器。使用 注解`@InverseMethod` ， 指明需要反过来转换所使用的转换方式。

```
public class Converter {
    @InverseMethod("stringToDate")
    public static String dateToString(long value) {
        // Converts long to String.
        return "" + value * 2;
    }

    public static long stringToDate(String value) {
        // Converts String to long.
        if(TextUtils.isEmpty(value)){
            return 0;
        }
        return Integer.parseInt(value);
    }
}
```

> 3-xml中使用转换器

```
<import type="com.hao.architecture.Converter"/>
<EditText
    android:id="@+id/birth_date"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@={Converter.dateToString(user.birthDate)}"
/>
```

系统内置的支持双向绑定的属性

1、系统内置的支持双向绑定的属性和其Adapter(可以去看源码)

Class	Attribute(s)	Binding adapter

Class	Attribute(s)	Binding adapter
AdapterView	android:selectedItemPosition android:selection	AdapterViewBindingAdapter
CalendarView	android:date	CalendarViewBindingAdapter
CompoundButton android:checked	CompoundButtonBindingAdapter	
DatePicker	android:year android:month android:day	DatePickerBindingAdapter
NumberPicker	android:value	NumberPickerBindingAdapter
RadioButton	android:checkedButton	RadioGroupBindingAdapter
RatingBar	android:rating	RatingBarBindingAdapter
SeekBar	android:progress	SeekBarBindingAdapter
TabHost	android:currentTab	TabHostBindingAdapter
TextView	android:text	TextViewBindingAdapter
TimePicker	android:hour android:minute	TimePickerBindingAdapter

知识扩展

@plurals

1、android中的Plurals（Quantity Strings）类型的作用

> 在不同语言中，可能出现单复数的情况，根据不同的数量，选择不同的语句是很重要的
> 例如: 1 device 和 2 devices 的区别

2、@plurals的语法

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals
        name="plural_name">
        <item
            quantity=["zero" | "one" | "two" | "few" | "many" | "other"]
            >text_string</item>
    </plurals>
</resources>
```

3、使用

> 1-xml中定义资源

```
<! --定义到资源文件即可 -->
<plurals name="subtitle_plural">
    <!--在使用时，可以根据数量来选择不同的字符串-->
    <!--还有zero、few等其它选项-->
    <item quantity="one">%s crime</item>
    <item quantity="other">%s crimes</item>
</plurals>
```

> 2-Java使用

```
String subtitle = getResources().getQuantityString(
        R.plurals.subtitle_plural,  // 1. 文本id
        1,   // 2. 数量，对应于"zero"、"two"等等
        1);  // 3. 填充占位符的内容
```

gradlew进行报错定位

1、在项目根目录(有gradlew)的目录下执行:

```
./gradlew compileDebugJavaWithJavac
```

> 输出信息：就定位到了目标真正错误的地方

```
F:\Project\FeatherLogin\app\src\main\java\com\hao\featherlogin\User.java:11: 错误: Entities and Pojos must h
......
......
```

问题汇总

参考资料

1. 官方文档: Data Binding Library