链滴

Android 官方文档翻译学习（Data Binding Library）

作者：angels
原文链接：https://ld246.com/article/1496279528425
来源网站：链滴

<h2 id="Data-Binding-Library-数据绑定库">Data Binding Library 数据绑定库</h2>

This document explains how to use the Data Binding Library to write declarative layouts nd minimize the glue code necessary to bind your application logic and layouts

本文介绍了如何使用数据绑定库去写声明布局文件和减少绑定你的应用程序的逻辑和布局所必需粘合代码。

The Data Binding Library offers both flexibility and broad compatibility — it's a support library, so you can use it with all Android platform versions back to Android 2.1 (API level 7+).

数据绑定库是一个支持库，它提供了灵活性和广阔的兼容性，所以你可以在版本 Android2.1 之的所有 android 平台中使用它。

To use data binding, Android Plugin for Gradle 1.5.0-alpha1 or higher is required. See hw to update the Android Plugin for Gradle.

使用数据绑定要求 Android 中的 Gradle 插件在 1.5.0-alpha1 或者更高的版本。

<h2 id="Build-Environment-构建环境">Build Environment 构建环境</h2>

To get started with Data Binding, download the library from the Support repository in the Android SDK manager.

开始使用数据绑定库前先要在 Android SDK 管理者中从支持库中下载该库。

To configure your app to use data binding, add the dataBinding element to your build.gradle file in the app module.

在你的应用程序的模块中的 build.gradle 文件中添加 dataBinding 元素去配置你的应用程序去用数据绑定库的功能。

Use the following code snippet to configure data binding:

使用下面的代码片段去配置使用数据绑定
<pre><code class="highlight-chroma">android {
  ....

  dataBinding {
      enabled = true
  }
}

</code></pre>

If you have an app module that depends on a library which uses data binding, your app odule must configure data binding in its build.gradle file as well.

如果你有一个应用程序模块依赖了一个使用数据绑定的库，你的应用程序模块同样需要在 build.g adle 文件中配置数据绑定。

Also, make sure you are using a compatible version of Android Studio. Android Studio 1.3 and later provides support for data binding as described in Android Studio Support for Data B nding.

此外，请确保你使用了一个兼容版本的 Android Studio。Android Studio 1.3 之后的都为数据 定提供了支持。

<h2 id="Data-Binding-Layout-Files-数据绑定布局文件">Data Binding Layout Files 数据绑定布 文件</h2>
<h3 id="Writing-your-first-set-of-data-binding-expressions">Writing your first set of data bi ding expressions</h3>
<h3 id="写你的第一组数据绑定表达式">写你的第一组数据绑定表达式</h3>

Data-binding layout files are slightly different and start with a root tag of layout followed by a data element and a view root element. This view element is what your root would be in a non-binding layout file. A sample file looks like this:

数据绑定布局文件略有不同，在布局开始的根标签后要跟着一个数据元素和一个根视图元素。你 根视图元素 jiang 将在一个不具约束力的不惧文件中。看起来就想下面的示例文件：
<pre><code class="highlight-chroma"><?xml version="1.0" encoding="utf-8"?>
<layout xmlns:a droid="http://schemas.android.com/apk/res/android">
  <data>
      <variable n me="user" type="com.example.User"/>
  </data>
  <LinearLayout
      android:orienta ion="vertical"
      android:layout width="match_parent"
      android:layout height="match_parent">

    <TextView a
droid:layout_width="wrap_content"
        android:layo
t_height="wrap_content"
        android:text
"@{user.firstName}"/>
    <TextView a
droid:layout_width="wrap_content"
        android:layo
t_height="wrap_content"
        android:text
"@{user.lastName}"/>
</LinearLayou
>
</layout>

</code></pre>

The user variable within data describes a property that may be used within this layout.</
>

使用数据变量描述一个可以在这个布局文件内使用的属性。

<pre><code class="highlight-chroma"><span class="highlight
cl"><variable name="user" type="com.example.User"/>

</code></pre>

Expressions within the layout are written in the attribute properties using the "@{}" syntax
Here, the TextView's text is set to the firstName property of user:
在布局文件内使用"@{}"语法写属性，在这里，TextView 的 text 设置为 user 的 firstName 属
。

<pre><code class="highlight-chroma"><span class="highlight
cl"><TextView android:layout_width="wrap_content"
        android:lay
ut_height="wrap_content"
        android:tex
="@{user.firstName}"/>

</code></pre>
<h2 id="Data-Object-数据对象">Data Object 数据对象</h2>

Let's assume for now that you have a plain-old Java object (POJO) for User:
让我们假设现在你有一个简单的 User Java 对象（POJO）

<pre><code class="highlight-chroma"><span class="highlight
cl">public class User {
  public final Stri
g firstName;
  public final Stri

g lastName;
    public User(Stri
g firstName, String lastName) {
        this.firstName
= firstName;
        this.lastName
= lastName;
    }
}

</code></pre>

This type of object has data that never changes. It is common in applications to have data that is read once and never changes thereafter. It is also possible to use a JavaBeans objects:<
li>
这种类型的对象有一个永远不会改变的数据。这种数据在应用程序中是常见的，这种数据一旦呗取一次就永远不会被改变。此外也可以使用 JavaBeans 对象。

<pre><code class="highlight-chroma"><span class="highlight
cl">public class User {
    private final Stri
g firstName;
    private final Stri
g lastName;
    public User(Stri
g firstName, String lastName) {
        this.firstName
= firstName;
        this.lastName
= lastName;
    }
    public String ge
FirstName() {
        return this.firs
Name;
    }
    public String ge
LastName() {
        return this.las
Name;
    }
}

</code></pre>

From the perspective of data binding, these two classes are equivalent. The expression @
user.firstName} used for the TextView's android:text attribute will access the firstName field in the former class and the getFirstName() method in the latter class. Alternatively, it will also be resolved to firstName() if that method exists.
从数据绑定的观点来看，这两个类是等价的。TextView 的 android：text 属性通过使用表达式{user.firsName}将获得模版类中的 firsName 字段和后面的类中的 getFirsName()方法。或者，它也以被解析为 firstName()，如果该方法存在。

<h2 id="Binding-Data-绑定数据">Binding Data 绑定数据</h2>

By default, a Binding class will be generated based on the name of the layout file, convert ng it to Pascal case and suffixing "Binding" to it. The above layout file was main_activity.xml s the generate class was MainActivityBinding. This class holds all the bindings from the layout roperties (e.g. the user variable) to the layout's Views and knows how to assign values for the binding expressions.The easiest means for creating the bindings is to do it while inflating:</p

默认的，将会根据布局文件的名字生成一个绑定类，将其转换为 Pascal 的实例并在名字后面添加 Binding 作为后缀.上述的布局文件是 main_activity.xml,所以生成类叫做 MainActivityBinding.该类 制来自布局的视图中的布局属性中的所有绑定并且知道如何去为绑定表达式分配值。在渲染布局时创 绑定是最简单的方式。

<pre><code class="highlight-chroma">@Override
protected void on reate(Bundle savedInstanceState) {
  super.onCreate(avedInstanceState);
  MainActivityBind ng binding = DataBindingUtil.setContentView(this, R.layout.main_activity);
  User user = new User("Test", "User");
  binding.setUser user);
}

</code></pre>

You're done! Run the application and you'll see Test User in the UI. Alternatively, you can get the view via:
你已经完成！运行应用程序你将在 UI 中看到测试的用户信息。或者，你可以通过视图其获得。</l >

<pre><code class="highlight-chroma">MainActivityBinding binding = MainActivityBinding.inflate(getLayoutInflater());

</code></pre>

If you are using data binding items inside a ListView or RecyclerView adapter, you may pr fer to use:

如果你正在使用数据绑定一个 ListView 或者 RecyclerView 适配器中的项目，你可能更喜欢使 ：

<pre><code class="highlight-chroma">ListItemBinding binding = ListItemBinding.inflate(layoutInflater, viewGroup, false);
//or

ListItemBinding binding = DataBindingUtil.inflate(layoutInflater, R.layout.list_item, viewGroup, false);

</code></pre>
<h2 id="Event-Handling-事件处理">Event Handling 事件处理</h2>

Data Binding allows you to write expressions handling events that are dispatched from the views (e.g. onClick). Event attribute names are governed by the name of the listener method with a few exceptions. For example, View.OnLongClickListener has a method onLongClick(), so the attribute for this event is android:onLongClick. There are two ways to handle an event.

数据绑定允许你写表达式去处理来自视图（点击）的转发的事件。通过监听方法宁子管理事件属名单，有些例外。例如，View.onLongClickListener 有一个 onLongCkick()方法，所以对于该事件属性是 android:onLongClick.以下是两个方式去处理事件。

Method References: In your expressions, you can reference methods that conform to the signature of the listener method. When an expression evaluates to a method reference, Data Binding wraps the method reference and owner object in a listener, and sets that listener on the target view. If the expression evaluates to null, Data Binding does not create a listener and set a null listener instead.

方法引用：在你的表达式中，你可以引用符合监听方法签名的方法。当一个表达式评估一个方法用时，数据绑定将方法的引用和监听器中拥有的对象包裹起来并且将监听器设置给目标对象。如果表式评估是 Null，数据绑定不会创建一个监听器而是设置一个值为 null 的监听器合取代

Listener Bindings: These are lambda expressions that are evaluated when the event happens. Data Binding always creates a listener, which it sets on the view. When the event is dispatched, the listener evaluates the lambda expression.

监听器绑定:当事件发生时，lambda 表达式将被评估。数据绑定总是会创建一个监听者，该监听会被设置到 view 上。当事件转发的时候，监听者评估该 kambda 表达式。

<h2 id="Method-References-方法引用">Method References 方法引用</h2>

Events can be bound to handler methods directly, similar to the way android: onClick can be assigned to a method in an Activity. One major advantage compared to the View#onClick attribute is that the expression is processed at compile time, so if the method does not exist or its signature is not correct, you receive a compile time error.

事件可以被直接绑定到处理方法上，与在一个 Actuvity 中使用 android：onClick 可以被分配一个方法上的方式类似。与设置 View#onClick 属性相比一个主要的优势是该表达式在编译时被处理如果该方法不存在或者它的签名不正确，你将收到一个编译时错误。

The major difference between Method References and Listener Bindings is that the actual listener implementation is created when the data is bound, not when the event is triggered. If you prefer to evaluate the expression when the event happens, you should use listener bindi g.

方法的引用和监听者的绑定这两种方式主要的不同是当数据绑定时被监听者的是被真实的创建，不是在事件触发时。如果你更倾向与在事件发生时去评估该表达式，你应该使用监听者绑定。

To assign an event to its handler, use a normal binding expression, with the value being t e method name to call. For example, if your data object has two methods:

使用值是作为方法名的正常的绑定表达式去分配一个事件给处理程序。例如，如果你的数据对象两个方法。
<pre><code class="highlight-chroma">public class MyHandlers {
  public void onCli kFriend(View view) { ... }
}

</code></pre>

The binding expression can assign the click listener for a View:

绑定表达式可以分配给视图的点击监听者:
<pre><code class="highlight-chroma"><?xml version="1.0" encoding="utf-8"?>
<layout xmlns:a droid="http://schemas.android.com/apk/res/android">
  <data>
      <variable n me="handlers" type="com.example.Handlers"/>
      <variable n me="user" type="com.example.User"/>
  </data>
  <LinearLayout
      android:orienta ion="vertical"
      android:layout width="match_parent"
      android:layout height="match_parent">
      <TextView a droid:layout_width="wrap_content"
          android:layo t_height="wrap_content"
          android:text "@{user.firstName}"
          android:onCl ck="@{handlers::onClickFriend}"/>

 </LinearLayou
>
</layout>

</code></pre>

Note that the signature of the method in the expression must exactly match the signatur
 of the method in the Listener object.

注意在表达式中方法的签名必须准确匹配监听者对象中方法的签名

<h2 id="Listener-Bindings-监听者绑定">Listener Bindings 监听者绑定</h2>

Listener Bindings are binding expressions that run when an event happens. They are simil
r to method references, but they let you run arbitrary data binding expressions. This feature is
available with Android Gradle Plugin for Gradle version 2.0 and later.

监听者的绑定是在一个事件发生时运行绑定的绑定表达式。他们类似方法的引用，但它们允许你
行任意的数据绑定表达式，该特性在使用 Android Gradle 插件且 Gradle 版本为 2.0 之后是可用的

In method references, the parameters of the method must match the parameters of the e
ent listener. In Listener Bindings, only your return value must match the expected return value
of the listener (unless it is expecting void). For example, you can have a presenter class that h
s the following method:

在方法的绑定中，方法的参数必须匹配事件监听者的参数。在监听者绑定中，只有你的返回值必
匹配监听者预期返回的值（除非它的愈切返回值是空）。例如，你可以有一个持久化的类，该类有以
方法：
<pre><code class="highlight-chroma"><span class="highlight
cl">public class Presenter {
  public void onSa
eClick(Task task){}
}

</code></pre>

Then you can bind the click event to your class as follows:

然后你可以使用如下代码绑定点时间到你的类。

<pre><code class="highlight-chroma"><span class="highlight
cl"> <?xml version="1.0" encoding="utf-8"?>
  <layout xmlns:

ndroid="http://schemas.android.com/apk/res/android">
  <data>
  <variable name="task" type="com.android.example.Task" />
  <variable name="presenter" type="com.android.example.Presenter" />
  </data>
  <LinearLayout android:layout_width="match_parent" android:layout_height="match_parent">
  <Button ndroid:layout_width="wrap_content" android:layout_height="wrap_content"
  android:onClick="@{() -> presenter.onSaveClick(task)}" />
  </LinearLayout>
</layout>

</code></pre>

Listeners are represented by lambda expressions that are allowed only as root elements o your expressions. When a callback is used in an expression, Data Binding automatically creats the necessary listener and registers for the event. When the view fires the event, Data Binding evaluates the given expression. As in regular binding expressions, you still get the null and hread safety of Data Binding while these listener expressions are being evaluated.

对于你的表达式，只有作为一个根元素，监听者才允许用 lambda 表达式表示。

Note that in the example above, we haven't defined the view parameter that is passed in o onClick(android.view.View). Listener bindings provide two choices for listener parameters: y u can either ignore all parameters to the method or name all of them. If you prefer to name t e parameters, you can use them in your expression. For example, the expression above could e written as:

注意在上面的例子中，我们还没有定义传递到 onClick（View view）中的视图参数。监听者绑 队友监听参入提供两种选择：你可以忽略所有方法的参数或者他们的名字。如果你更倾向于命名参数 你可以在你的表达式中使用他们。例如，以上的表达式可以被写作：
<pre><code class="highlight-chroma"> android:onClick="@{(view) -> presenter.onSaveClick(task)}"

</code></pre>

Or if you wanted to use the parameter in the expression, it could work as follows:

或者如果你想在你表达式中去使用参数，它可以作为如下方式运行：
<pre><code class="highlight-chroma"> public class Presenter {
  public void onaveClick(View view, Task task){}

  }
  android:onClick=
@{(theView) -> presenter.onSaveClick(theView, task)}"

</code></pre>

You can use a lambda expression with more than one parameter:

超过一个参数时你可以使用一个 lambad 表达式

<pre><code class="highlight-chroma"><span class="highlight
cl">public class Presenter {
    public void on
ompletedChanged(Task task, boolean completed){}
}

</code></pre>
<pre><code class="highlight-chroma"><span class="highlight
cl"> <CheckBox android:layout_width="wrap_content" android:layout_height="wrap_cont
nt"
        android:onC
eckedChanged="@{(cb, isChecked) -> presenter.completeChanged(task, isChecked)}" /&gt

</code></pre>

If the event you are listening to returns a value whose type is not void, your expressions
ust return the same type of value as well. For example, if you want to listen for the long click
vent, your expression should return boolean.

如果你监听的事件返回了一个类型不是 void 的值，你的表达式必须返回一样类型的值。例如，
果你想去监听长点击事件，你的表达式应该返回布尔类型。
<pre><code class="highlight-chroma"><span class="highlight
cl">public class Presenter {
  public boolean o
LongClick(View view, Task task){}
}

</code></pre>
<pre><code class="highlight-chroma"><span class="highlight
cl">android:onLongClick="@{(theView) -> presenter.onLongClick(theView, task)}"

</code></pre>

If the expression cannot be evaluated due to null objects, Data Binding returns the defaul
 Java value for that type. For example, null for reference types, 0 for int, false for boolean, etc.
/li>
如果表达式因为空对象不能被评估，数不绑定返回一个默认类型的 Java 值。例如，Null 对应引
类型，0 对应整型，false 对应布尔类型。

If you need to use an expression with a predicate (e.g. ternary), you can use void as a sy
bol.

如果你需要在表达式中使用断言，你可以使用 void 作为一个标志。
<pre><code class="highlight-chroma"><span class="highlight
cl">android:onClick="@{(v) -> v.isVisible() ? doSomething() : void}"

</code></pre>

<h2 id="Avoid-Complex-Listeners-避开复合监听">Avoid Complex Listeners 避开复合监听</h2

<hr>

Listener expressions are very powerful and can make your code very easy to read. On the
other hand, listeners containing complex expressions make your layouts hard to read and un
aintainable. These expressions should be as simple as passing available data from your UI to
our callback method. You should implement any business logic inside the callback method th
t you invoked from the listener expression.

监听表达式是非常强大的，它可以提高你的代码的可读性。另一方面，监听者包含的复合表达式
加你的布局的阅读难度和不可维护。对于你的回调方法的表达式应该尽可能简单就像通过你的 UI 的
递可用的数据给你的回调方法一样。你应该在从监听表达式调用的回调方法中事件所有的业务逻辑。<
p>

Some specialized click event handlers exist and they need an attribute other than android
onClick to avoid a conflict. The following attributes have been created to avoid such conflicts

存在一些专门的单击事件处理程序需要比 android: onClick 多一个其他的属性以避免冲突。以
的属性被创建去避免冲突：

<pre><code class="highlight-chroma"><span class="highlight
cl">Class Listener Setter Attribute
SearchView setO
nSearchClickListener(View.OnClickListener) android:onSearchClick
ZoomControls setO
tOnZoomInClickListener(View.OnClickListener) android:onZoomIn
ZoomControls setO
tOnZoomOutClickListener(View.OnClickListener) android:onZoomOut

</code></pre>
<h2 id="Layout-Details-布局详情">Layout Details 布局详情</h2>
<h3 id="Imports">Imports</h3>

Zero or more import elements may be used inside the data element. These allow easy ref
rence to classes inside your layout file, just like in Java.

零个或者更多被导入的元素可以在数据元素里面使用。在你的布局文件里可以就像在 Java 里 面
样很容易的引用类。

<pre><code class="highlight-chroma"><span class="highlight
cl"><data>
  <import typ
="android.view.View"/>
</data>

</code></pre>

Now, View may be used within your binding expression:

现在，可以在你的绑定表达式里面使用 View

<pre><code class="highlight-chroma"><span class="highlight
cl"><TextView
  android:text="@
user.lastName}"
  android:layout_
idth="wrap_content"
  android:layout_
eight="wrap_content"
  android:visibilit
="@{user.isAdult ? View.VISIBLE : View.GONE}"/>

</code></pre>

When there are class name conflicts, one of the classes may be renamed to an "alias:"</
>

当类名冲突时，一个类可以使用别名。

<pre><code class="highlight-chroma"><span class="highlight
cl"><import type="android.view.View"/>
<import type="
om.example.real.estate.View"
  alias="Vista"
>

</code></pre>

Now, Vista may be used to reference the com.example.real.estate.View and View may be sed to reference android.view.View within the layout file. Imported types may be used as type references in variables and expressions:

现在，在布局文件理 Vista 可以被使用去引用 com.example.real.estate.View 并且 View 也可被使用去引用 com.example.real.estate.View。在变凉和表达式中导入类型可以被作为类型引用去使。
<pre><code class="highlight-chroma"><data>
  <import type= com.example.User"/>
  <import type= java.util.List"/>
  <variable nam ="user" type="User"/>
  <variable nam ="userList" type="List<User>"/>
</data>

</code></pre>

Note: Android Studio does not yet handle imports so the autocomplete for imported vari bles may not work in your IDE. Your application will still compile fine and you can work aroun the IDE issue by using fully qualified names in your variable definitions.

注意：Android Studio 至今不能处理导入，所以在你的 IDE 中自动导入变量不能工作。您的应程序仍然可以很好地编译，你可以通过在变量定义使用完全合格的名称解决此问题的 IDE。

<pre><code class="highlight-chroma"><TextView
  android:text="@ ((User)(user.connection)).lastName}"
  android:layout_ idth="wrap_content"
  android:layout_ eight="wrap_content"/>

</code></pre>

Imported types may also be used when referencing static fields and methods in expressi ns:

当用引用静态字段和方法时，也可以在表达式中使用导入的类型。
<pre><code class="highlight-chroma"><data>
  <import type= com.example.MyStringUtils"/>

  <variable nam
="user" type="com.example.User"/>
</data>
...
<TextView
 android:text="@
MyStringUtils.capitalize(user.lastName)}"
  android:layout_w
dth="wrap_content"
  android:layout_he
ght="wrap_content"/>

</code></pre>

Just as in Java, java.lang.* is imported automatically.

就像在 Java 中一样，java.lang.*是自动导入的
<h2 id="Variables">Variables</h2>

Any number of variable elements may be used inside the data element. Each variable ele
ent describes a property that may be set on the layout to be used in binding expressions with
n the layout file.

任何变量元素的成员都可以在数据元素里面使用。每一个变量元素描述一个属性，该属性可以在
局文件的布局里通过使用绑定表达式设置。
<pre><code class="highlight-chroma"><span class="highlight
cl"><data>
  <import type=
android.graphics.drawable.Drawable"/>
  <variable nam
="user" type="com.example.User"/>
  <variable nam
="image" type="Drawable"/>
  <variable nam
="note" type="String"/>
</data>

</code></pre>

The variable types are inspected at compile time, so if a variable implements Observable
r is an observable collection, that should be reflected in the type. If the variable is a base class
or interface that does not implement the Observable* interface, the variables will not be obse
ved!

变量类型在编译时检查，所以如果一个变量实现了 Observable 或者是一个 obsevable 集合，
类型中该变量应该被反射。如果该变凉是一个基本类型或者没有实现 Observable 及其子类接口的接
，该变量就不能被观察！

When there are different layout files for various configurations (e.g. landscape or portrait) the variables will be combined. There must not be conflicting variable definitions between th se layout files.

当在不同的布局文件中且各个布局文件的配置也不同时，该变凉将被整合。在那些布局文件之间量的定义绝对不能有冲突。

The generated binding class will have a setter and getter for each of the described variab es. The variables will take the default Java values until the setter is called — null for reference ypes, 0 for int, false for boolean, etc.

生成的绑定类对于每一个描述的变量将会生成一个 setter 和 getter 构造器。在 setter 方法被调之前该变量将获得一个默认的 Java 值 - 引用是 null，整型是 0，布尔类型是 false

A special variable named context is generated for use in binding expressions as needed. he value for context is the Context from the root View's getContext(). The context variable will be overridden by an explicit variable declaration with that name.

为了使用绑定表达式中，生成一个特殊的被成为上下文的变量是必须的。该上下文的值就是来自根视图中通过 getContext（）方法获得的上下文。该上下文变量将被一个使用名字显示声明的变量盖。

<h2 id="Custom-Binding-Class-Names-自定义绑定类名字">Custom Binding Class Names 自义绑定类名字</h2>

By default, a Binding class is generated based on the name of the layout file, starting it wi h upper-case, removing underscores (_) and capitalizing the following letter and then suffixi g "Binding". This class will be placed in a databinding package under the module package. For example, the layout file contact_item.xml will generate ContactItemBinding. If the module pac age is com.example.my.app, then it will be placed in com.example.my.app.databinding.

默认的，一个绑定类是基于布局文件的名字进行生成，使用大写开始，移除下划线（_）并且后跟着的字大写然后添加 Binding 后缀。该类奖杯放置在模块包下的 databinding 包中。例如，布局件 contact_item.xml 将生成 ContactItemBinding.如果模块包是 com.example.myapp，那么该类被放置在 com.example.myapp.databinding 包中。

Binding classes may be renamed or placed in different packages by adjusting the class att ibute of the data element. For example:

绑定类的名字可以通过调整数据元素中类的属性可以重命名。例如：
<pre><code class="highlight-chroma"><data class="ContactItem">
  ...

</data>

</code></pre>

This generates the binding class as ContactItem in the databinding package in the modul package. If the class should be generated in a different package within the module package, i may be prefixed with ".":

生成的 ContactItem 绑定类再模块包下的 databinding 包中，如果希望该类在模块包下不同的中，可以使用.前缀。
<pre><code class="highlight-chroma"><data class=".ContactItem">
  ...
</data>

</code></pre>

In this case, ContactItem is generated in the module package directly. Any package may e used if the full package is provided:

在这种情况下，ContactItem 直接生成在模块包下，如果提供完整的包，任何包都可以使用它。/p>

<pre><code class="highlight-chroma"><data class="com.example.ContactItem">
    ...
</data>

</code></pre>
<h2 id="Includes">Includes</h2>

Variables may be passed into an included layout's binding from the containing layout by using the application namespace and the variable name in an attribute:

来自于包含布局的变量可以在一个属性中通过使用应用程序命名空间和和变量名传入一个被 inclued 进来的绑定布局中
<pre><code class="highlight-chroma"><?xml version="1.0" encoding="utf-8"?>
<layout xmlns:a droid="http://schemas.android.com/apk/res/android"
    xmlns:bind="h tp://schemas.android.com/apk/res-auto">
  <data>
    <variable n me="user" type="com.example.User"/>
  </data>
  <LinearLayout

```
</span></span><span class="highlight-line"><span class="highlight-cl">    android:orientation="vertical"
</span></span><span class="highlight-line"><span class="highlight-cl">    android:layout width="match_parent"
</span></span><span class="highlight-line"><span class="highlight-cl">    android:layout height="match_parent"&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">    &lt;include layout="@layout/name"
</span></span><span class="highlight-line"><span class="highlight-cl">        bind:user="@{user}"/&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">    &lt;include layout="@layout/contact"
</span></span><span class="highlight-line"><span class="highlight-cl">        bind:user="@{user}"/&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">&lt;/LinearLayou&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">&lt;/layout&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span></code></pre>
</li>
<li>
<p>Here, there must be a user variable in both the name.xml and contact.xml layout files.</>
</li>
<li>
<p>这里的 name.xml 和 contact.xml 布局文件中都必须有一个 user 变量</p>
</li>
<li>
<p>Data binding does not support include as a direct child of a merge element. For example, the following layout is not supported:</p>
</li>
<li>
<p>数据绑定不支持在作为一个 merge 元素的直接子元素进行 include 操作。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl">&lt;?xml version="1.0" encoding="utf-8"?&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">&lt;layout xmlns:android="http://schemas.android.com/apk/res/android"
</span></span><span class="highlight-line"><span class="highlight-cl">    xmlns:bind="http://schemas.android.com/apk/res-auto"&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">  &lt;data&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">    &lt;variable name="user" type="com.example.User"/&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">  &lt;/data&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">  &lt;merge&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">    &lt;include layout="@layout/name"
</span></span><span class="highlight-line"><span class="highlight-cl">        bind:user="@{user}"/&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">    &lt;include layout="@layout/contact"
</span></span><span class="highlight-line"><span class="highlight-cl">        bind:user="@{user}"/&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">  &lt;/merge&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">&lt;/layout&gt;
```

</code></pre>
<h2 id="Expression-Language-表达式语言">Expression Language 表达式语言</h2>
<h3 id="Common-Features-常用特性">Common Features 常用特性</h3>

The expression language looks a lot like a Java expression. These are the same:

表达式语言看上去很像 java 表达式。下面是一些相同的部分

<blockquote>

Mathematical + - / * % 数学运算

String concatenation + 字符串连接

Logical && || 逻辑运算符

Binary & | ^ 位运算符

Unary + - ! ~ 一元运算符

Shift >> >>> << 位移运算

Comparison == > < >= <= 比较运算

instanceof 判断实例类型

Grouping () 分组

Literals - character, String, numeric, null

Cast

Method calls

Field access

Array access []

Ternary operator ?:
<h3 id="Examples--例子">Examples: 例子</h3>

</blockquote>
<pre><code class="highlight-chroma"><span class="highlight
cl">android:text="@{String.valueOf(index + 1)}"
android:visibility=
@{age < 13 ? View.GONE : View.VISIBLE}"
android:transitio
Name='@{"image_" + id}'

</code></pre>
<h2 id="Missing-Operations-没有的操作">Missing Operations 没有的操作</h2>

A few operations are missing from the expression syntax that you can use in Java.
一些可以在 Java 中使用的操作在表达式语法中是没有的

<blockquote>

this
super
new
Explicit generic invocation

</blockquote>
<h2 id="Null-Coalescing-Operator-Null-的聚合操作">Null Coalescing Operator Null 的聚合操
</h2>
<hr>
The null coalescing operator (??) chooses the left operand if it is not null or the right if it i
null.
null 聚合操作(??)如果不为 null 选择左边的操作数，如果为 null，选择右边的操作数
<pre><code class="highlight-chroma"><span class="highlight
cl">android:text="@{user.displayName ?? user.lastName}"

</code></pre>

This is functionally equivalent to:

以下的功能是等价的
<pre><code class="highlight-chroma"><span class="highlight
cl">android:text="@{user.displayName != null ? user.displayName : user.lastName}"

</code></pre>
#Property Reference 属性引用

The first was already discussed in the Writing your first data binding expressions above: s
ort form JavaBean references. When an expression references a property on a class, it uses the

same format for fields, getters, and ObservableFields.

在上面有关 JavaBean 的使用部分中已经讨论了如何写你的第一个数据绑定表达式。当表达式引了一个类的属性时，他要使用一样的 fields，getters，ObservableFields

<pre><code class="highlight-chroma">android:text="@{user.lastName}"

</code></pre>
#Avoiding NullPointerException 避免空指针异常

Generated data binding code automatically checks for nulls and avoid null pointer exceptions. For example, in the expression @{user.name}, if user is null, user.name will be assigned its default value (null). If you were referencing user.age, where age is an int, then it would default to 0.

生成的数据绑定代码会自动检查空引用和空指针异常。例如，在表达式 <code>@{user.name}<code>,如果 user 是 null，user.name 将被配置一个默认的 null 值。如果你引用 user.age,age 是一 int 类型字段，那么他将默认是 0.

#Collections 集合

Common collections: arrays, lists, sparse lists, and maps, may be accessed using the [] operator for convenience.
<pre><code class="highlight-chroma"><data>
  <import type= android.util.SparseArray"/>
  <import type= java.util.Map"/>
  <import type= java.util.List"/>
  <variable nam ="list" type="List<String>"/>
  <variable nam ="sparse" type="SparseArray<String>"/>
  <variable nam ="map" type="Map<String, String>"/>
  <variable nam ="index" type="int"/>
  <variable nam ="key" type="String"/>
</data>
...
android:text="@{list[index]}"
...
android:text="@{

parse[index]}"
...
android:text="@{
ap[key]}"

</code></pre>
#String Literals

When using single quotes around the attribute value, it is easy to use double quotes in t
e expression:
<pre><code class="highlight-chroma"><span class="highlight
cl">android:text='@{map["firstName"]}'

</code></pre>

It is also possible to use double quotes to surround the attribute value. When doing so, S
ring literals should either use the ' or back quote (`).
<pre><code class="language-mysql highlight-chroma">android
text="@{map[`firstName`]}"
<span cla
s="highlight-w"><android<span class="highlight
p">:text=<
pan class="highlight-s2">"@{map['firstName']}"
<span cla
s="highlight-w">
</code></pre>
#Resources

It is possible to access resources as part of expressions using the normal syntax:

使用正常的表达式语法可以获得资源
<pre><code class="highlight-chroma"><span class="highlight
cl">android:padding="@{large? @dimen/largePadding : @dimen/smallPadding}"

</code></pre>

Format strings and plurals may be evaluated by providing parameters:

格式化字符串和复数可以通过提供的参数进行评估：
<pre><code class="highlight-chroma"><span class="highlight
cl">android:text="@{@string/nameFormat(firstName, lastName)}"
android:text="@
@plurals/banana(bananaCount)}"

</code></pre>

When a plural takes multiple parameters, all parameters should be passed:

当一个复数采用多个参数是，所有的参数都应该被传入。

<pre><code class="highlight-chroma"><span class="highlight
cl">
  Have an orange
  Have %d orange

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

</code></pre>

Some resources require explicit type evaluation.
一些资源要求明确评估类型

<pre><code class="highlight-chroma">
<span class="highlight
cl">Type Normal Reference Expression Reference
String[] @a
ray @stringArray
int[] @ar
ay @intArray
TypedArray
@array @typedArray
Animator
animator @animator
StateListAnimator
 @animator @stateListAnimator
color int @
olor @color
ColorStateList
@color @colorStateList

</code></pre>
<h2 id="Data-Objects">Data Objects</h2>

Any plain old Java object (POJO) may be used for data binding, but modifying a POJO will
not cause the UI to update. The real power of data binding can be used by giving your data o
jects the ability to notify when data changes. There are three different data change notificatio
 mechanisms, Observable objects, observable fields, and observable collections.

数据绑定可以使用所有的 Java 对象（POJO），但是修改一个 POJO 不会引起一个 UI 的更新。
过给予你的数据对象一个在数据发生变化时可以通知的能力来使用数据绑定真正强大的能力。这是三
不同的数据改变通知机制，Observable 对象，observable 字段，observable 集合


```html
<p>When one of these observable data object is bound to the UI and a property of the data
bject changes, the UI will be updated automatically.</p>
</li>
<li>
<p>当被绑定到 UI 中的有一个是 observable 数据对象时，如果数据对象的属性值改变了，UI 也会
动更新。</p>
</li>
</ul>
<h2 id="Observable-Objects">Observable Objects</h2>
<ul>
<li>A class implementing the Observable interface will allow the binding to attach a single lis
ener to a bound object to listen for changes of all properties on that object.</li>
<li>一个实现了 Observable 接口的类将允许给绑定对象绑定关联一个监听器去监听该对象中所有属
的变化情况。</li>
<li>The Observable interface has a mechanism to add and remove listeners, but notifying is
p to the developer. To make development easier, a base class, BaseObservable, was created to
implement the listener registration mechanism. The data class implementer is still responsible
for notifying when the properties change. This is done by assigning a Bindable annotation to
he getter and notifying in the setter.</li>
<li>Observable 接口有一个添加和移除监听器的机制，而已通知是在不断的发展改进的。创建一个
本的 BaseObservable 类区实现监听器注册机制对于开发来说是很容易的。通过分配一个 Bindable
解给数据类的 getter 和 setter，当属性改变时，数据类的视线仍然能响应通知。</li>
</ul>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">private static class User extends BaseObservable {
</span></span><span class="highlight-line"><span class="highlight-cl">    private String fir
tName;
</span></span><span class="highlight-line"><span class="highlight-cl">    private String las
Name;
</span></span><span class="highlight-line"><span class="highlight-cl">    @Bindable
</span></span><span class="highlight-line"><span class="highlight-cl">    public String ge
FirstName() {
</span></span><span class="highlight-line"><span class="highlight-cl">        return this.firs
Name;
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    @Bindable
</span></span><span class="highlight-line"><span class="highlight-cl">    public String ge
LastName() {
</span></span><span class="highlight-line"><span class="highlight-cl">        return this.las
Name;
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    public void setFi
stName(String firstName) {
</span></span><span class="highlight-line"><span class="highlight-cl">        this.firstName
= firstName;
</span></span><span class="highlight-line"><span class="highlight-cl">        notifyPropert
Changed(BR.firstName);
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    public void setL
stName(String lastName) {
</span></span><span class="highlight-line"><span class="highlight-cl">        this.lastName
= lastName;
</span></span><span class="highlight-line"><span class="highlight-cl">        notifyPropert
Changed(BR.lastName);
```

  }
}

</code></pre>

The Bindable annotation generates an entry in the BR class file during compilation. The B class file will be generated in the module package. If the base class for data classes cannot be changed, the Observable interface may be implemented using the convenient PropertyChang Registry to store and notify listeners efficiently.

在编译的时候，Bindable 注解在 BR 类文件中生成一个条目。在模块包下生成一个 BR 文件。如数据类是一个不能改变的类，Observable 接口会通过使用适当的 PropertyChangeRegister 去有效存储和通知监听者来实现。

#ObservableFields

A little work is involved in creating Observable classes, so developers who want to save t me or have few properties may use ObservableField and its siblings ObservableBoolean, Obse vableByte, ObservableChar, ObservableShort, ObservableInt, ObservableLong, ObservableFloa , ObservableDouble, and ObservableParcelable. ObservableFields are self-contained observab e objects that have a single field. The primitive versions avoid boxing and unboxing during ac ess operations. To use, create a public final field in the data class:
在创建 Observable 类时还是要做一些工作的，所以一些想去解约事件或者希望有一些可使用的性的开发着可以使用 ObservableField 和与之类似的 ObservableBoolean，ObservableByte，Obse vableChar，ObservableDouble，ObservableLong，ObservableFloat，ObservableDouble，Ob ervableParcelable。ObservableFields 是一个有单一字段的自包含的 observable 对象。最初的版是避免在操作时做装箱和拆箱操作。在数据类中以 public final 字段形式创建并使用。

<pre><code class="highlight-chroma">private static class User {
  public final Obs rvableField<String> firstName =
    new Observab eField<>();
  public final Obs rvableField<String> lastName =
    new Observab eField<>();
  public final Obs rvableInt age = new ObservableInt();
}

</code></pre>

That's it! To access the value, use the set and get accessor methods:
就是这样，使用 set 和 get 方法去获取值

<pre><code class="highlight-chroma">

user.firstName.set("Google");

int age = user.age get();

</code></pre>
<h2 id="Observable-Collections">Observable Collections</h2>

Some applications use more dynamic structures to hold data. Observable collections allo
 keyed access to these data objects. ObservableArrayMap is useful when the key is a reference
type, such as String.

一些应用更多地使用动态结构去控制数据。Observable collections 允许键值对的形势去获取数
对象。ObservableArrayMap 是很常用的一个，它的 key 是一个引用类型，比如 String。

<pre><code class="highlight-chroma"><span class="highlight
cl">ObservableArrayMap<String, Object> user = new ObservableArrayMap<>();
user.put("firstNa
e", "Google");
user.put("lastNam
", "Inc.");
user.put("age", 17)

</code></pre>

In the layout, the map may be accessed through the String keys:

在布局中，可以通过 String 类型的 key 访问 map。

<pre><code class="highlight-chroma"><data>
    <import typ
="android.databinding.ObservableMap"/>
    <variable na
e="user" type="ObservableMap<String, Object>"/>
</data>
...
<TextView
    android:text='@
user["lastName"]}'
    android:layout_
idth="wrap_content"
    android:layout_
eight="wrap_content"/>
<TextView
    android:text='@
String.valueOf(1 + (Integer)user["age"])}'
    android:layout_
idth="wrap_content"
    android:layout_
eight="wrap_content"/>

</code></pre>

ObservableArrayList is useful when the key is an integer:

ObservableArrayList 也是很常用的，它的 key 是一个 integer：
<pre><code class="highlight-chroma"><span class="highlight
cl">ObservableArrayList<Object> user = new ObservableArrayList<>();
user.add("Google"
;
user.add("Inc.");
user.add(17);

</code></pre>

In the layout, the list may be accessed through the indices:

在布局中可以通过 下标访问 list

<pre><code class="highlight-chroma"><data>
  <import typ
="android.databinding.ObservableList"/>
  <import typ
="com.example.my.app.Fields"/>
  <variable na
e="user" type="ObservableList<Object>"/>
</data>
...
<TextView
  android:text='@
user[Fields.LAST_NAME]}'
  android:layout_
idth="wrap_content"
  android:layout_
eight="wrap_content"/>
<TextView
  android:text='@
String.valueOf(1 + (Integer)user[Fields.AGE])}'
  android:layout_
idth="wrap_content"
  android:layout_
eight="wrap_content"/>

</code></pre>
<h2 id="Generated-Binding">Generated Binding</h2>

The generated binding class links the layout variables with the Views within the layout. As

discussed earlier, the name and package of the Binding may be customized. The Generated b nding classes all extend ViewDataBinding.

使用 Views 内部的 layout 可以将生成的绑定类于布局变量链接起来。就像前面讨论的一样，绑 的包名可疑被自定义。所有生成的绑定类都继承 ViewDataBinding 这个类

<h3 id="Creating">Creating</h3>

The binding should be created soon after inflation to ensure that the View hierarchy is no disturbed prior to binding to the Views with expressions within the layout. There are a few w ys to bind to a layout. The most common is to use the static methods on the Binding class.The inflate method inflates the View hierarchy and binds to it all it one step. There is a simpler vers on that only takes a LayoutInflater and one that takes a ViewGroup as well:
在 View 被 inflation 后应该尽快创建绑定，这样能确保在 layout 里用表达式绑定 View 之前 Vi w 的层级不被干扰。以下是几种绑定 layout 的方式。最常用的是在绑定类上使用静态方法 inflate 方 去 Inflates View 的层级和绑定它，所有操作只需要一步完成。这是 i 个简单版本的方法，里面只使 一噶 LayoutInflater 和一个 ViewGroup：

<pre><code class="highlight-chroma">MyLayoutBinding binding = MyLayoutBinding.inflate(layoutInflater);
MyLayoutBinding binding = MyLayoutBinding.inflate(layoutInflater, viewGroup, false);

</code></pre>

If the layout was inflated using a different mechanism, it may be bound separately:

如果 layout 是通过不同的机制进行 inflated，那么它可以被单独地绑定。
<pre><code class="highlight-chroma">MyLayoutBinding binding = MyLayoutBinding.bind(viewRoot);

</code></pre>

Sometimes the binding cannot be known in advance. In such cases, the binding can be cr ated using the DataBindingUtil class:

有时，我们并不能预先知道绑定的东西。在这种情况下，可以使用 DataBindingUtils 类去创建 定。

<pre><code class="highlight-chroma">ViewDataBinding binding = DataBindingUtil.inflate(LayoutInflater, layoutId,
    parent, attachT Parent);
ViewDataBinding inding = DataBindingUtil.bindTo(viewRoot, layoutId);

</code></pre>

<h2 id="Views-With-IDs">Views With IDs</h2>

A public final field will be generated for each View with an ID in the layout. The binding oes a single pass on the View hierarchy, extracting the Views with IDs. This mechanism can be faster than calling findViewById for several Views. For example:

在 layout 中使用 View 的一个 ID 为每个 View 生成一个 public final 类型的字段。在 View 的级中，绑定根据 IDs 提取 View 并简单的传入 View 的层级中。这个机制比通过调用各种 Views 的 fidViewById 方法获取 View 更加的快速。例如：

<pre><code class="highlight-chroma"><layout xmlns:android="http://schemas.android.com/apk/res/android">
  <data>
    <variable nme="user" type="com.example.User"/>
  </data>
  <LinearLayout
    android:orienation="vertical"
    android:layou_width="match_parent"
    android:layou_height="match_parent">
    <TextViendroid:layout_width="wrap_content"
        android:layut_height="wrap_content"
        android:tex="@{user.firstName}"
  android:id="@+d/firstName"/>
    <TextViendroid:layout_width="wrap_content"
        android:layut_height="wrap_content"
        android:tex="@{user.lastName}"
  android:id="@+d/lastName"/>
  </LinearLayou>
</layout>

</code></pre>

Will generate a binding class with:

将会使用如下代码生成一个绑定类：

<pre><code class="highlight-chroma"><span class="highlight
cl">public final TextView firstName;
public final TextVi
w lastName;

</code></pre>

IDs are not nearly as necessary as without data binding, but there are still some instances where access to Views are still necessary from code.

如果没有数据绑定，IDs 并不是必须的，但是仍然有些情况下需要从代码里面访问 Views 时，IDs 仍然是需要的。

<h2 id="Variables-变量">Variables 变量</h2>

Each variable will be given accessor methods.
每一个变量都将被给予一个存储器方法

<pre><code class="highlight-chroma"><span class="highlight
cl"><data>
  <import typ
="android.graphics.drawable.Drawable"/>
  <variable na
e="user" type="com.example.User"/>
  <variable na
e="image" type="Drawable"/>
  <variable na
e="note" type="String"/>
</data>

</code></pre>

will generate setters and getters in the binding:
在绑定时将会生成一个 setters 和一个 getters 存储器

<pre><code class="highlight-chroma"><span class="highlight
cl">public abstract com.example.User getUser();
public abstract vo
d setUser(com.example.User user);
public abstract Dr
wable getImage();
public abstract vo
d setImage(Drawable image);
public abstract Str
ng getNote();
public abstract vo
d setNote(String note);

</code></pre>
<h2 id="ViewStubs">ViewStubs</h2>

ViewStubs are a little different from normal Views. They start off invisible and when they ither are made visible or are explicitly told to inflate, they replace themselves in the layout by nflating another layout.

ViewStubs 与一般的 Views 有些许不同。ViewStubs 开始时一般时不可见的，当 ViewStubs 置为可见的活着明确调用 inflate，ViewStubs 将会使用其他的 layout 替换本身。

Because the ViewStub essentially disappears from the View hierarchy, the View in the bin ing object must also disappear to allow collection. Because the Views are final, a ViewStubPro y object takes the place of the ViewStub, giving the developer access to the ViewStub when it exists and also access to the inflated View hierarchy when the ViewStub has been inflated.</ >

因为从 View 的层级中 ViewStub 本身并不会被渲染出来，绑定对象中的 View 对于 collection 必须允许其消失。因为这个 Views 是 final 的，当 ViewStub 已经被 inflated 且在 ViewStubProxy 象取代 ViewStub 的位置时，如果获得了 View 的层级的 ViewStub 是存的，开发者便能获得该 View tub。

When inflating another layout, a binding must be established for the new layout. Therefo e, the ViewStubProxy must listen to the ViewStub's ViewStub.OnInflateListener and establish he binding at that time. Since only one can exist, the ViewStubProxy allows the developer to s t an OnInflateListener on it that it will call after establishing the binding.

在 inflating 其他的 layout 时，对于一个新的 layout，一个绑定必须是明确的。因此，ViewStu Proxy 必须监听 ViewStubs 的 ViewStub.OnInflateListener 并且明确这绑定的时间。ViewStubProx 只允许开发者去设置一个 OnInflateListener，在明确地绑定时，该监听者将被调用。

<h2 id="Advanced-Binding-高级绑定">Advanced Binding 高级绑定</h2>
<h3 id="Dynamic-Variables-动态变量">Dynamic Variables 动态变量</h3>

At times, the specific binding class won't be known. For example, a RecyclerView.Adapter operating against arbitrary layouts won't know the specific binding class. It still must assign t e binding value during the onBindViewHolder(VH, int).

有时，特殊的绑定类不能被了解。例如，一个 RecyclerView.Adapter 的操作违反了任意布局不知道指定的绑定类。当 onBindViewHolder（VH，int）回调时它仍然能够被分配到绑定的值。

In this example, all layouts that the RecyclerView binds to have an "item" variable. The Bi dingHolder has a getBinding method returning the ViewDataBinding base.

比如以下例子，RecyclerView 的所有布局绑定一个 item 变量。BindinngHolder 又一个 getBin ing 方法，该方法返回基本的 ViewDataBinding。

<pre><code class="highlight-chroma"><span class="highlight
cl">public void onBindViewHolder(BindingHolder holder, int position) {
    final T item = mI
ems.get(position);
    holder.getBindi
g().setVariable(BR.item, item);
    holder.getBindi
g().executePendingBindings();
}

</code></pre>
<h3 id="Immediate-Binding-立即绑定">Immediate Binding 立即绑定</h3>

When a variable or observable changes, the binding will be scheduled to change before t
e next frame. There are times, however, when binding must be executed immediately. To forc
 execution, use the executePendingBindings() method.
当一个变量活着 observable 变化时，在下一帧前绑定操作将被调度去改变。然而，有时绑定必
被立即调用，可以使用 executePendingBindings()方法。

<h3 id="Background-Thread-后台线程">Background Thread 后台线程</h3>

You can change your data model in a background thread as long as it is not a collection.
ata binding will localize each variable / field while evaluating to avoid any concurrency issues.
/li>
你也可以在尽可能长耗时的后台线程去改变你的数据模型，该模型不是一个 collection.当产生避
任何并发行问题的评估操作时，数据绑定将使每一个变量 / 字段局域化。

<h2 id="Attribute-Setters-属性设置">Attribute Setters 属性设置</h2>

Whenever a bound value changes, the generated binding class must call a setter method
on the View with the binding expression. The data binding framework has ways to customize
which method to call to set the value.
每当一个绑定值发生改变时，使用绑定表达式的 View 对应生成的绑定类都必须调用 setter 方法
数据绑定框架有多种方式去自定义被调用去设置改变的值的方法。

<h2 id="Automatic-Setters-自动设置">Automatic Setters 自动设置</h2>

For an attribute, data binding tries to find the method setAttribute. The namespace for t
e attribute does not matter, only the attribute name itself.

对于一个属性，数据绑定尝试着去查找 setAttribute 的方法。于属性的命名空间没有关系，只
属性本身的名字有关。

For example, an expression associated with TextView's attribute android:text will look for
 setText(String). If the expression returns an int, data binding will search for a setText(int) met
od. Be careful to have the expression return the correct type, casting if necessary. Note that d
ta binding will work even if no attribute exists with the given name. You can then easily "creat
" attributes for any setter by using data binding. For example, support DrawerLayout doesn't
ave any attributes, but plenty of setters. You can use the automatic setters to use one of these

例如，一个与 TextView 的 android : text 关联的表达式就会查找一个 setText(String) 的方法如果该表达式返回一个 int 值，数据绑定会继续搜索一个 setText(int)的方法。如果有需要，要特别意表达式返回正确类型。

<pre><code class="highlight-chroma"><android.support.v4.widget.DrawerLayout
    android:layout width="wrap_content"
    android:layout_ eight="wrap_content"
    app:scrimColor "@{@color/scrim}"
    app:drawerList ner="@{fragment.drawerListener}"/>

</code></pre>
<h3 id="Renamed-Setters-重命名Setters">Renamed Setters 重命名 Setters</h3>

Some attributes have setters that don't match by name. For these methods, an attribute ay be associated with the setter through BindingMethods annotation. This must be associated with a class and contains BindingMethod annotations, one for each renamed method. For ex mple, the android:tint attribute is really associated with setImageTintList(ColorStateList), not s tTint.
一些属性的 setters 不能通过名字进行匹配。对于此种情况，我们可以使用 BindingMethods 注将一个属性和 setter 进行关联。对于每个重命名的方法，该方法必须属于一个类并且包含 BindingMe hod 注解。例如，android:tint 属性实际上是鱼 setImageTintList（ColorStateList）方法相关联，不是 setTint 方法。

<pre><code class="highlight-chroma">@BindingMethods({
    @BindingMe hod(type = "android.widget.ImageView",
                attri ute = "android:tint",
                met od = "setImageTintList"),
})

</code></pre>

It is unlikely that developers will need to rename setters; the android framework attribute have already been implemented.
Android 框架的属性早已经被实现，开发者去重命名框架的方法是不可能的。

<h3 id="Custom-Setters-自定义setters">Custom Setters 自定义 setters</h3>

Some attributes need custom binding logic. For example, there is no associated setter for the android:paddingLeft attribute. Instead, setPadding(left, top, right, bottom) exists. A static inding adapter method with the BindingAdapter annotation allows the developer to customiz

how a setter for an attribute is called.

一些属性需要自定义绑定逻辑。例如，对于 android：paddingLeft 属性是与 setter 没有联系。而是与 setPadding(left,top,right,bottom)方法相关联。使用 BindingAdapter 注解的一个静态绑适配器的方法允许开发者去定义该属性的 setter 方法被调用时做些什么。

The android attributes have already had BindingAdapters created. For example, here is th one for paddingLeft:

android 的属性早已被 BindingAdaoters 创建。例如，这是一个 paddingLeft 的例子：

<pre><code class="highlight-chroma">@BindingAdapter("android:paddingLeft")
public static void etPaddingLeft(View view, int padding) {
    view.setPadding padding,
                view.g tPaddingTop(),
                view.g tPaddingRight(),
                view.g tPaddingBottom());
}

</code></pre>

Binding adapters are useful for other types of customization. For example, a custom load r can be called off-thread to load an image.

绑定适配器对于自定义的类型是很有用的。例如，一个自定的一加载器可以调用离线的线程去加一个图像。

Developer-created binding adapters will override the data binding default adapters when there is a conflict.

当有冲突的时候，被开发者创建的绑定适配器将被默认的绑数据绑定适配器覆盖。

You can also have adapters that receive multiple parameters.

你也可以有一个接收多个参数的适配器

<pre><code class="highlight-chroma"><span class="highlight

cl">@BindingAdapter({"bind:imageUrl", "bind:error"})
public static void
oadImage(ImageView view, String url, Drawable error) {
    Picasso.with(vie
.getContext()).load(url).error(error).into(view);
}

</code></pre>
<pre><code class="highlight-chroma"><span class="highlight
cl"><ImageView app:imageUrl="@{venue.imageUrl}"
app:error="@{@d
awable/venueError}"/>

</code></pre>
This adapter will be called if both imageUrl and error are used for an ImageView and ima
eUrl is a string and error is a drawable.
如果被一个 ImageView 使用的 imageUrl 错误或者 imageUrl 是一个错误的 drawable 字符串
该适配器都会被调用

Custom namespaces are ignored during matching.

自定义命名空间在匹配的过程中被忽略

You can also write adapters for android namespace.

你也可以对 android 命名空间写一个适配器

Binding adapter methods may optionally take the old values in their handlers. A method
aking old and new values should have all old values for the attributes come first, followed by
he new values:
在他们的处理程序中，绑定适配器的方法可以随意的获得旧的值。一个方法获得旧的值和新的值
时候，首先获得的是该属性的所有旧的值，然后才是新的值。
<pre><code class="highlight-chroma"><span class="highlight
cl">
@BindingAdapter
"android:paddingLeft")
public static void
etPaddingLeft(View view, int oldPadding, int newPadding) {
    if (oldPadding !=
newPadding) {
        view.setPaddi
g(newPadding,
                view
getPaddingTop(),
                view
getPaddingRight(),
                view
getPaddingBottom());
    }

}

</code></pre>

Event handlers may only be used with interfaces or abstract classes with one abstract me
hod. For example:
事件处理程序仅可以和接口或者抽象类的一个抽象方法一起使用。例如：

<pre><code class="highlight-chroma"><span class="highlight
cl">@BindingAdapter("android:onLayoutChange")
public static void
etOnLayoutChangeListener(View view, View.OnLayoutChangeListener oldValue,
        View.OnLayou
ChangeListener newValue) {
    if (Build.VERSI
N.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (oldValue !
 null) {
            view.remo
eOnLayoutChangeListener(oldValue);
        }
        if (newValue
= null) {
            view.addO
LayoutChangeListener(newValue);
        }
    }
}

</code></pre>

When a listener has multiple methods, it must be split into multiple listeners. For example
 View.OnAttachStateChangeListener has two methods: onViewAttachedToWindow() and onVi
wDetachedFromWindow(). We must then create two interfaces to differentiate the attributes
nd handlers for them.
当一个监听者有多个方法时，它必须被分割成多个简体器。例如，View.OnAttachStateChangeL
stener 有两个方法：onViewAttachedToWindow（）和 onViewDetachdFromWindow().我们必
为该属性创建两个不同的接口并处理它们。

<pre><code class="highlight-chroma"><span class="highlight
cl">@TargetApi(VERSION_CODES.HONEYCOMB_MR1)
public interface O
ViewDetachedFromWindow {
    void onViewDe
achedFromWindow(View v);
}

@TargetApi(VERS
ON_CODES.HONEYCOMB_MR1)
public interface O
ViewAttachedToWindow {
    void onViewAtt
chedToWindow(View v);
}

```
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span></code></pre>
<ul>
<li>Because changing one listener will also affect the other, we must have three different bin
ing adapters, one for each attribute and one for both, should they both be set.</li>
<li>对于每一个应该被设置的属性，我们必须有多个不同的绑定适配器，因为改变一个监听器也会影
另外一个，</li>
</ul>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">@BindingAdapter("android:onViewAttachedToWindow")
</span></span><span class="highlight-line"><span class="highlight-cl">public static void
etListener(View view, OnViewAttachedToWindow attached) {
</span></span><span class="highlight-line"><span class="highlight-cl">    setListener(view
 null, attached);
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">@BindingAdapter
"android:onViewDetachedFromWindow")
</span></span><span class="highlight-line"><span class="highlight-cl">public static void
etListener(View view, OnViewDetachedFromWindow detached) {
</span></span><span class="highlight-line"><span class="highlight-cl">    setListener(view
 detached, null);
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">@BindingAdapter(
"android:onViewDetachedFromWindow", "android:onViewAttachedToWindow"})
</span></span><span class="highlight-line"><span class="highlight-cl">public static void
etListener(View view, final OnViewDetachedFromWindow detach,
</span></span><span class="highlight-line"><span class="highlight-cl">        final OnView
ttachedToWindow attach) {
</span></span><span class="highlight-line"><span class="highlight-cl">    if (VERSION.SD
_INT &gt;= VERSION_CODES.HONEYCOMB_MR1) {
</span></span><span class="highlight-line"><span class="highlight-cl">        final OnAttac
StateChangeListener newListener;
</span></span><span class="highlight-line"><span class="highlight-cl">        if (detach ==
null &amp;amp;&amp;amp; attach == null) {
</span></span><span class="highlight-line"><span class="highlight-cl">            newListene
 = null;
</span></span><span class="highlight-line"><span class="highlight-cl">        } else {
</span></span><span class="highlight-line"><span class="highlight-cl">            newListene
 = new OnAttachStateChangeListener() {
</span></span><span class="highlight-line"><span class="highlight-cl">                @Overr
de
</span></span><span class="highlight-line"><span class="highlight-cl">                public v
id onViewAttachedToWindow(View v) {
</span></span><span class="highlight-line"><span class="highlight-cl">                    if (att
ch != null) {
</span></span><span class="highlight-line"><span class="highlight-cl">                        att
ch.onViewAttachedToWindow(v);
</span></span><span class="highlight-line"><span class="highlight-cl">                    }
</span></span><span class="highlight-line"><span class="highlight-cl">                }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                @Overr
de
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">            public v
id onViewDetachedFromWindow(View v) {
</span></span><span class="highlight-line"><span class="highlight-cl">                if (det
ch != null) {
</span></span><span class="highlight-line"><span class="highlight-cl">                    det
ch.onViewDetachedFromWindow(v);
</span></span><span class="highlight-line"><span class="highlight-cl">                }
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">        };
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    final OnAttac
StateChangeListener oldListener = ListenerUtil.trackListener(view,
</span></span><span class="highlight-line"><span class="highlight-cl">            newList
ner, R.id.onAttachStateChangeListener);
</span></span><span class="highlight-line"><span class="highlight-cl">    if (oldListener
!= null) {
</span></span><span class="highlight-line"><span class="highlight-cl">        view.remo
eOnAttachStateChangeListener(oldListener);
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    if (newListen
r != null) {
</span></span><span class="highlight-line"><span class="highlight-cl">        view.addO
AttachStateChangeListener(newListener);
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span></code></pre>
<ul>
<li>
<p>The above example is slightly more complicated than normal because View uses add and
emove for the listener instead of a set method for View.OnAttachStateChangeListener. The a
droid.databinding.adapters.ListenerUtil class helps keep track of the previous listeners so that
hey may be removed in the Binding Adaper.</p>
</li>
<li>
<p>以上的例子比一般的例子要稍微复杂。因为 View 对监听器使用 add 和 remove 去替换一个 Vie
。OnAttachStateChangeListener 的 set 方法。android.databinding.adapters.ListenerUtil 类帮
跟踪之前的简体器一遍监听器可以在 Binding Adapter 中移除。</p>
</li>
<li>
<p>By annotating the interfaces OnViewDetachedFromWindow and OnViewAttachedToWin
ow with @TargetApi(VERSION_CODES.HONEYCOMB_MR1), the data binding code generator
nows that the listener should only be generated when running on Honeycomb MR1 and new
evices, the same version supported by addOnAttachStateChangeListener(View.OnAttachState
hangeListener).</p>
</li>
<li>
<p>OnViewDetachedFromWindow 和 OnViewAttachedToWindsw 接口通过使用注解 @Taeget
pi（VERSION_CODES.HONEYCOMEB_MR1），只有当运行在 Honeycomb MR1 盒心得设备时，
据绑定代码生产者才会生成监听器。</p>
</li>
</ul>
<hr>
```

<h2 id="Converters-转换器">Converters 转换器</h2>
<h3 id="Object-Conversions-对象转换器">Object Conversions 对象转换器</h3>

When an Object is returned from a binding expression, a setter will be chosen from the a
tomatic, renamed, and custom setters. The Object will be cast to a parameter type of the cho
en setter.

当从一个绑定表达式中返回一个对象，该对象的一个自动的、从命名的河自定义的 setters 将被
择。该对象的类型将被转换成被选择的 setter 的参数类型。

This is a convenience for those using ObservableMaps to hold data. for example:

使用 ObservableMaps 去控制数据的这是很方便的，例如：

<pre><code class="highlight-chroma"><span class="highlight
cl"><TextView
  android:text='@
userMap["lastName"]}'
  android:layout_
idth="wrap_content"
  android:layout_
eight="wrap_content"/>

</code></pre>

The userMap returns an Object and that Object will be automatically cast to parameter ty
e found in the setter setText(CharSequence). When there may be confusion about the parame
er type, the developer will need to cast in the expression.
userMap 返回一个对象并且该对象将被自动转换成在 setText（CharSequence）方法中找到的
数类型。当参数类型可能被混淆的时候，开发者奖需要去转换表达式。

<h3 id="Custom-Conversions-自定义转换器">Custom Conversions 自定义转换器</h3>

Sometimes conversions should be automatic between specific types. For example, when
etting the background:
一些时候的转换器应该自动选择特定的类型。例如，当设置背景时：

<pre><code class="highlight-chroma"><span class="highlight
cl"><View
  android:backgr
und="@{isError ? @color/red : @color/white}"
  android:layout_
idth="wrap_content"
  android:layout_
eight="wrap_content"/>

</code></pre>

Here, the background takes a Drawable, but the color is an integer. Whenever a Drawable

is expected and an integer is returned, the int should be converted to a ColorDrawable. This c
nversion is done using a static method with a BindingConversion annotation:
这里，北京获得了一个 Drawable，但是颜色是一个整型。无论是一个 Drawable 异常还是返回
个整型，该整型都应该被转换成一个 ColorDrawable。使用一个使用了 BindingConversion 注解的
态方法完成了转换。

<pre><code class="highlight-chroma"><span class="highlight
cl">@BindingConversion
public static Colo
Drawable convertColorToDrawable(int color) {
  return new Colo
Drawable(color);
}

</code></pre>

Note that conversions only happen at the setter level, so it is not allowed to mix types like
this:
注意，转换旨在 setter 方法里触发，所以转换不允许像下面这样的混合类型。

<pre><code class="highlight-chroma"><span class="highlight
cl"><View
  android:backgr
und="@{isError ? @drawable/error : @color/white}"
  android:layout_
idth="wrap_content"
  android:layout_
eight="wrap_content"/>

</code></pre>
###Android Studio Support for Data Binding Android Studio 对于数据绑定的支持。
Android Studio supports many of the code editing features for data binding code. For ex
mple, it supports the following features for data binding expressions:
Android Studio 支持很多语句数据绑定代码的代码编辑特性。例如，它支持对于数据绑定表达
的一下特性：

Syntax highlighting
语法高亮
Flagging of expression language syntax errors
XML code completion
References, including navigation (such as navigate to a declaration) and quick documenta
ion

Note: Arrays and a generic type, such as the Observable class, might display errors when
here are no errors.
注意：Arrays 和一般的类型，例如 Observable 类没有错误的时候可能会显示错误。

The Preview pane displays default values for data binding expressions if provided. In the
ollowing example excerpt of an element from a layout XML file, the Preview pane displays the
PLACEHOLDER default text value in the TextView.
如果支持的话，预览版对于数据绑定表达式会展示默认的值。以下例子类子引用自一个来自于布局
xml 文件的元素，预览版会显示 TextView 的
PLACEHOLDER 的迷人值

<pre><code class="highlight-chroma"><TextView android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{user.firstName, default=PLACEHOLDER}"/>

</code></pre>

If you need to display a default value during the design phase of your project, you can also use tools attributes instead of default expression values, as described in Designtime Layout Attributes.
在设计你的项目的时候，如果你需要去现实一个默认值，虽然在设计时的布局属性中描述了，你可以使用 tools 属性替换默认的表达式值，

转载请注明原文链接：Android 官方文档翻译学习（Data Binding Library）