# 译文版：

你听到的消息是真实的：第一步：在Android中使用[Data Binding](https://developer.android.com/tools/data-binding/guide.html)。第二步：profit。Yigit Boyar和George Mount是帮助开发Android数据绑定库的Google开发人员，帮助开发人员以最小的努力建立丰富且响应迅速的用户体验。在[Bay Area Android Dev Group的](http://www.meetup.com/bayareaandroid/)这次演讲中，他们演示了如何使用Data Bindings通过删除数据驱动的UI的样板来改进您的应用程序，从而使您能够编写更清晰，更好的代码。

学点英语：

俗语：straight from the horse's mouth。 Straight from the horse's mouth，从字面上来解释它的意思是：直接来自马的嘴巴。可是，这个俗语的真正意思是：你听到的消息是真实的，因为这消息来自可靠的来源。马和消息可靠又有什么关系呢？你要知道，远在人们还没有使用汽车的年代，人们是依靠马作为他们主要的交通工具，认为马是十分可靠的。现在，汽车已经早就代替了马，但是这个俗语仍然被广为使用。现在我们再来举个例子：

I tell you I know for certain that Lisa will be promoted to manager. The boss told me himself, so it's straight from the horse's mouth!

这个人说：“告诉你，我知道莉沙肯定会被提升为经理的。这是老板亲口告诉我的，这个消息是绝对可靠的！”

### 简介[（0:00）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,0,true);)

我们是George Mount和Yigit Boyar，我们在Android UI工具包团队工作。我们有很多关于数据绑定的信息可以与您分享，还有大量的代码可供您使用。我们将讨论数据绑定如何工作的重要方面，如何将其整合到您的应用程序中，以及如何与其他组件协同工作，以及我们将提及一些最佳实践。

### 为什么数据绑定？[（0:44）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,1,true);)

您可能想知道为什么我们决定实施这个库。这是一个常见用例的例子。

<LinearLayout …>

<TextView android:id="@+id/name"/>

<TextView android:id="@+id/lastName"/></LinearLayout>

这是您随时可以看到的Android用户界面。假设你有一堆带有ID的视频。你的设计师对你说，“好吧，让我们尝试添加新的信息到这个布局”，所以当你添加任何视频时，你需要粘贴另一个ID。返回到Java代码，以修改UI。

**private** TextView mName**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

mName **=** **(**TextView**)** findViewById**(**R**.**id**.**name**);}public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

mName**.**setText**(null);**

**}** **else** **{**

mName**.**setText**(**user**.**getName**());**

**}}**

你编写一个新的TextView，你可以从UI中找到它，然后设置你的逻辑，这样当你需要更新你的用户时，你必须在TextView上设置对应信息。

**private** TextView mName**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

mName **=** **(**TextView**)** findViewById**(**R**.**id**.**name**);**

mLastName **=** **(**TextView**)** findViewById**(**R**.**id**.**lastName**);}public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

mName**.**setText**(null);**

mLastName**.**setText**(null);**

**}** **else** **{**

mName**.**setText**(**user**.**getName**());**

mLastName**.**setText**(**user**.**getLastName**());**

**}}**

总而言之，仅仅是为了向用户界面添加一个view，你需要做很多事情。这似乎是很愚蠢的样板代码，不需要任何的智力。

已经有一些非常好的库来使这个更容易和更可靠。例如，如果你使用ButterKnife，你可以得到两个丑陋的viewByIds，使得它阅读起来更容易。你可以摆脱额外的代码，告诉ButterKnife删除它。

**private** TextView mName**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

ButterKnife**.**bind**(this);}public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

mName**.**setText**(null);**

mLastName**.**setText**(null);**

**}** **else** **{**

mName**.**setText**(**user**.**getName**());**

mLastName**.**setText**(**user**.**getLastName**());**

**}}**

这是一个很大的进步，但我们可以更进一步。我们可以说：“好吧，为什么我需要为这些创建项目？有些框架就可以产生它。我有一个布局文件，我有ID。”所以你可以使用Holdr。它帮助你处理你的文件，然后为它们创建视图。您从Holdr启动，将您输入的字段转换为字段名称。

**private** Holdr\_ActivityMain holder**;protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

holder **=** **new** Holdr\_ActivityMain**(**findViewById**(**content**));}public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

holder**.**name**.**setText**(null);**

holder**.**lastName**.**setText**(null);**

**}** **else** **{**

holder**.**name**.**setText**(**user**.**getName**());**

holder**.**lastName**.**setText**(**user**.**getLastName**());**

**}}**

这个要更好一点，但在代码中还是有一些不必要的东西。有很大一部分代码，是我们没有用过的，我们也无法减少代码量。这个也是非常简单的代码：我有一个用户对象，我只是想把这个对象里面的数据移到视图类中。当你看到这样的代码时，你有多少次犯了错误？你记得改变一件事，但忘记改变另一件事，并最终导致生产崩溃。这是我们想要关注的部分：我们想要跳过所有的样板代码。

当你使用 Data Binding时，和使用Holdr非常相似，但是你可以做很少的工作。Data Binding自己来规定其余部分。

**private** ActivityMainBinding mBinding**;protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

mBinding **=** DataBindingUtil**.**setContentView**(this,**

R**.**layout**.**activity\_main**);}public** **void** **updateUI(**User user**)** **{**

mBinding**.**setUser**(**user**);}**

### 幕后[（3:53）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,15,true);)

Data Binding如何在幕后工作？看看之前的布局文件：

<LinearLayout …>

<TextView android:id="@id/name" />

<TextView android:id="@id/lastName" /></LinearLayout>

我有这些ID，但为什么我需要它们，如果我能找到他们在我的Java代码，我其实不需要他们，所以我可以摆脱他们。在他们的位置，我放上我最想要展示的东西。

<LinearLayout …>

<TextView android:text="@{user.name}"/>

<TextView android:text="@{user.lastName}"/></LinearLayout>

现在，当我看这个布局文件时，我知道TextView显示的是什么。它变得非常明显，所以我不需要回头去阅读我的Java代码。我们设计的数据绑定库的方式不包含任何不易解释的方法。如果您在布局文件中使用某些东西，则需要告诉Data Binding它是什么。你只是简单地说：“我们用这种类型的user来标记这个布局文件，现在我们要找到它了。”如果你的设计师要求你添加另一个视图，你只需添加一行再显示你的新视图，没有其他代码需要更改。

<layout>

<data>

<variable name="user"

type="com.android.example.User"/>

</data>

<LinearLayout …>

<TextView android:text="@{user.name}"/>

<TextView android:text="@{user.lastName}"/>

<TextView android:text='@{"" + user.age}'/>

</LinearLayout></layout>

发现错误也很容易。你可以看看上面的代码，然后说：“哦，看！空字符串加user.age！“你只需在整数上设置文本，然后砰！我们做了很多次，只是发生了。

### 但它是如何工作的？[（5:57）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,19,true);)

数据绑定库的第一件事是处理您的布局文件。通过“进程”，我的意思是当你的应用程序正在被编译时，它会进入你的布局文件，找到关于数据绑定的所有东西，抓取这些信息并删除它。我们删除它，因为视图系统不知道它，所以它消失。

第二步是通过语法来解析这些表达式。例如，在这种情况下：

<TextView android:visibility="@user.isAdmin ? View.VISIBLE : View.GONE}"/>

这user是一个ID，这View是一个ID，另一个View是一个ID。它们就像真实物体一样是标识符，但是我们现在还不知道它们到底是什么。其他的事情是看不见的或消失的。有场访问，整个事情是一个三元组。这就是我们迄今为止所了解的。我们从文件中解析事物，并理解里面的内容。

第三步是解决你的代码正在编译时发生的依赖关系。例如，在这个步骤中，我们看一下user.isAdmin并找出它的含义。我们认为：“好的，这个方法在这个用户类中变成了一个布尔值。我知道这个表达式在运行时意味着某种布尔值。“

最后一步是编写数据绑定器。我们编写你不需要写的类。总之，最后一步：利润

### 一个案例[（7:40）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,27,true);)

这是一个布局文件的实际案例。

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

<data>

<variable name="user" type="com.android.example.User"/>

</data>

<RelativeLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView android:text="@{user.name}"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<TextView android:text="@{user.lastname}"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

</RelativeLayout></layout>

在我们处理的过程中，我们摆脱了视图系统不知道的所有事情，将它们链接起来，然后放回我们的绑定标签：

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView android:tag="binding\_1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<TextView android:tag="binding\_2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

</RelativeLayout>

这实际上是我们如何使数据绑定向后兼容。当你把它放在一个姜饼装置，这个可怜的家伙不知道发生了什么事情。

### Expression Tree[（8:01）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,30,true);)

<TextView android:text="@{user.age < 18 ? @string/redacted : user.name}"/>

这是另一个表达式的例子。当我们解析这个，它变成了一个表达式树，在编译时解决。注意到它在编译时发生是很重要的，所以当应用程序开始运行时，你已经知道了一切。我们检查这个表达式的左边，这是一个布尔值。我们检查右边，这是一个字符串。资源也是一个字符串。所以我有一个布尔，字符串，字符串，三元，这也是一个字符串。有一个文本属性，我有一个字符串。我如何设置？

有一个完美的setText(CharSequence)。现在，数据绑定知道如何将该表达式转换为Java代码。如果详细说明，那就是TextView和ImageView。

<TextView android:text="@{myVariable}"/>

textView.setText(myVariable);<ImageView android:src="@{user.image}"/>

imageView.setSrc(user.image);

ImageView是一个源属性，所以它是正确的，在上面的例子中，使用setSrc？不，因为ImageView上没有设置源方法。相反，这里面有一个ImageView的源代码方法。但是Data Binding怎么知道这个呢？

这就是所谓的源属性，因为你习惯于使用该属性，数据绑定必须支持它。

<TextView …/>

textView.setText(myVariable);<ImageView android:src="@{user.image}"/>

imageView.setImageResource(user.image);

@BindingMethod(

type = android.widget.ImageView.class,

attribute = "android:src",

method = "setImageResource")

我们创建了这些注释，在这里您可以简单地说：“好吧，在ImageView类中，将源图映射到这个方法”。我们只写了一次，实际上我们构建了一次框架。我们提供它，但是你可能有你想添加的自定义视图。一旦你添加该方法，数据绑定知道如何解决这个问题。再次，这一切都发生在编译时间。

### Data Binding好的地方[（9:54）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,36,true);)

数据绑定使您的生活变得更加轻松。我们来看看我们支持的表达式语言，大部分是 Java。它允许像字段访问，方法调用，参数，添加，比较，索引访问数组，常量访问，甚至三元表达式的东西。这基本上就是你想要的Java表达式。还有一些不做的事情，比如说new。我们真的不希望你做new你的表达。

我们的基本目标是使您的表达式尽可能****简短和可读，****就像在XML中一样。我们不希望您只为了访问您的联系人姓名而编写超长表达式。我们希望你能够使用contact.name。我们看着它，想：“好吧，这是一个领域，还是一个吸气？”或者你可以有一个“名字”的方法。

我们也做自动的空检查，这实际上真的很酷。如果你想访问这个名字，但是联系方式是空的，那么写在脖子上会有多少痛苦contact null ? null : contact.friend null ? :？你不想这样做。现在，如果contact为null，则整个表达式为null。

我们也有空合并运算符，您可能已经从其他语言中看到过。这只是一个简单的方法来做这个三元运算符：

contact**.**lastName **??** contact**.**namecontact**.**lastName **!=** **null** **?** contact**.**lastName **:** contact**.**name

它说，如果第一个不是空的，选择第一个。如果为空，则选择第二个。

我们也有使用括号操作符的列表访问和地图访问。如果你有contacts[0]，那个联系人可以是一个列表或一个数组，那就没事了。如果您有contactInfo，则可以使用括号表示法。这有点容易。

### Resources[（12:20）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,48,true);)

我们希望您能够在表达式中使用资源。Android中没有资源的表达式语言会是什么？现在，您可以在表达式中正确使用资源和字符串格式。

在表达式中：

**android:**padding**=**"@{isBig ? @dimen/bigPadding : @dimen/smallPadding}"

内联字符串格式：

**android:**text**=**"@{@string/nameFormat(firstName, lastName)}"

内联复数：

**android:**text**=**"@{@plurals/banana(bananaCount)}"

### Automagic属性[（13:00）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,49,true);)

这里我们有一个DrawerLayout ...

<android.support.v4.widget.DrawerLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

app:scrimColor="@{@color/scrim}"/>

drawerLayout**.**setScrimColor**(**

resources**.**getColor**(**R**.**color**.**scrim**))**

我们有这个属性app:scrimColor。DrawerLayout上没有稀松平常的颜色，但碰巧有setScrimColor。setScrimColor当我们有一个名称的属性，我们寻找这个scrimColor，我们检查是否类型匹配。首先我们看颜色，这是一个int。如果setScrimColor拿一个int，这是一个匹配。很方便！

### 事件处理[（13:41）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,50,true);)

我不知道你们中有多少人clicked使用过按钮或视图，但我们也在这里支持数据绑定。你可以使用clicked，但是现在也支持任何事件。当然，这可以回到华而不实的东西。你甚至可以做一些事情，你把一个任意的事件处理程序作为表达式的一部分（我不是说我建议你这样做，但是你可以！）。你也可以做一些奇怪的听众，比如onTextChanged。TextWatcher上有三个方法，但是大家只关心onTextChanged，对吧？你可以实际访问其中的一个，如果你想，或所有的人。

<Button android:onClick="clicked" …/><Button android:onClick="@{handlers.clicked}" …/> <Button android:onClick="@{isAdult ? handlers.adultClick : handlers.childClick}" …/> <Button android:onTextChanged="@{handlers.textChanged}" …/>

### Observability详情[（14:56）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,51,true);)

当你更新你的视图会发生什么？想象一下，我们有一家商店，而且我们有一个价格最近已经变化的商品。这必须自动更新我们的用户界面。这是怎么发生的？有了数据绑定，那真的很便宜，而且容易。

我们要做的第一件事是创建一个项目，一些可观察的对象。在这里，我扩展了基本的可观察对象，然后我们在那里有我们的领域。

**public** **class** **Item** **extends** BaseObservable **{**

**private** String price**;**

**@Bindable**

**public** String **getPrice()** **{**

**return** **this.**name**;**

**}**

**public** **void** **setPrice(**String price**)** **{**

**this.**price **=** price**;**

notifyPropertyChanged**(**BR**.**price**);**

**}}**

我们通过在此添加通知它notifyPropertyChanged。但是，我们通知什么会改变？我们必须在@Bindable诠释上加注getPrice。这就产生了这个BR.priceBR类中的价格字段。BR就像R类，我们只是为你生成它，它只是吸收这些绑定资源。但是，您可能不希望我们侵犯您的整个层次，所以我们也允许您实现可观察类。是的，我听到了一个与一个人抱怨...在这里，我们允许你自己实现它。

**public** **class** **Item** **implements** Observable **{**

**private** PropertyChangeRegistry callbacks **=** **new** …

…

**@Override**

**public** **void** **addOnPropertyChangedCallback(**

OnPropertyChangedCallback callback**)** **{**

callbacks**.**add**(**callback**);**

**}**

**@Override**

**public** **void** **removeOnPropertyChangedCallback(**

OnPropertyChangedCallback callback**)** **{**

callbacks**.**remove**(**callback**);**

**}}**

我们有这样一个方便的类，PropertyChangedRegistry它可以让你基本上接受这些回调并通知他们。你们中的一些人可能会认为这只是一个痛苦的脖子，而不是想要有一个可观察的领域。从本质上讲，每一个都是一个可观察的对象，它只有一个条目。它的设置很方便，当你使用accessImage的时候，它实际上是访问那个图片中的内容。如果您访问价格，它将访问该价格的字符串内容。

关于这些对象的特殊之处在于，在Java代码中，您必须调用set和get方法，但是在绑定表达式中，您可以说item.price，我们将知道我们需要调用getter。所以当价格变化时，它只是设定价值。

**public** **class** **Item** **{**

**public** **final** ObservableField**<**Drawable**>** image **=**

**new** ObservableField**<>();**

**public** **final** ObservableField**<**String**>** price **=**

**new** ObservableField**<>();**

**public** **final** ObservableInt inventory **=**

**new** **ObservableInt();}**item**.**price**.**set**(**"$33.41"**);**

在其他情况下，你可能会有更多的“blobby”数据。这通常发生在开发周期的开始阶段，特别是在原型开发阶段，您可能会遇到一些从网络上掉落的blob，而您并不想真正定义对象，因此您可能需要一张地图。在这种情况下，所有你需要做的就是有一个可观察的地图，你可以把你的项目，然后你可以访问它们。不幸的是，你不能以同样的方式访问它：你必须使用括号表示法。

ObservableMap**<**String**,** Object**>** item **=**

**new** ObservableArrayMap**<>();**item**.**put**(**"price"**,** "$33.41"**);**

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text='@{item["price"]}'/>

### Notify on Any Thread[（18:29）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,62,true);)

这里的一个便利是你不必在UI线程上通知：你可以更新任何你想要的线程。但是，请注意，我们将在UI线程上阅读，所以您必须小心。另外，请不要这样做与列表！对于列表，你仍然应该在UI线程上通知，因为我们要在UI线程上读取，而且我们需要UI线程上的长度，我们不做任何类型的同步。你可能已经知道从Recycler和ListView，这非常不安。这是因为列表，而不是因为这些类。

### Performance[（19:21）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,63,true);)

也许这个项目最重要的考虑是不要让它变慢。数据绑定因为速度慢而臭名昭着，所以在Android中，我们仔细考虑了这一点，我们相信我们做得很好。

表现最重要的是基本上是零反射。一切都在编译时发生。有时，事情是不方便的，因为它发生在编译时间，但从长远来看，我们不在乎。我们不想在应用程序运行时解决任何问题。

第二部分是你可以免费得到的东西。假设您在布局中使用“数据绑定”（Data Binding）来命名对象的价格，但是对象的价格会发生变化。所以新的价格来了，通知来了。数据绑定只会更新TextView，没有别的，但TextView将被测量。如果你手工编写代码，那么你不太可能写这个代码，而只是再次启动这个视图。所以这带来了一个免费的好处。

数据绑定中的另一个性能优势来自于以下两个表达式：

<TextView android:text="@{user.address.street}"/><TextView android:text="@{user.address.city}"/>

你有user.address另一个user.address。代码DataBinding将生成如下所示：

Address address **=** user**.**getAddress**();**String street **=** address**.**getStreet**();**String city **=** address**.**getCity**();**

它将把地址移动到一个局部变量，然后对其进行操作。现在想象一下，有一些计算，实际上是昂贵的。数据绑定只会做一次。这是另一件你不会亲手做的事情。

表演的另一个积极的副作用是findById。当你findById在Android的视图上编码时，它实际上是给所有的孩子看的，像“孩子零，你能通过ID找到这个视图吗？”这个孩子问孩子，然后去下一个孩子等等直到你找到视图。然后，你findViewById再次编码另一个视图，同样的事情再次发生。

但是，初始化数据绑定时，我们实际上知道在编译时我们感兴趣的是哪些视图，所以我们有一种方法来查找所有我们想要的视图。我们遍历一次布局层次来收集所有的视图。这是同样的故事，我们遍历它，但它只发生一次。第二次我们需要另一种观点，没有第二遍，因为我们已经找到了所有的观点。

性能是关于一点细节。你在你的代码中包含一个库，所以是的，一些行为会改变，是的，会有一些成本。但是，所有这些事情，我想我们都是这样做的，甚至有时比你写的代码更好，这是非常重要的。

### RecyclerView和Data Binding[（22:14）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,71,true);)

使用ViewHolders对于ListViews非常常见，RecyclerView强制执行这种模式。如果您查看数据绑定生成的内容，您将看到它实际上为您生成了ViewHolder。它有领域，它知道意见。您也可以轻松使用RecyclerView的内部。我们创建一个ViewHolder，它具有这个基本的创建方法，一个静态的方法，告诉UserItemBinding（从用户项目布局文件生成的类）。所以你调用UserItemBinding膨胀。而现在你有一个非常简单的ViewHolder类，它只保持对生成的绑定的引用，绑定方法就是这样的。

**public** **class** **UserViewHolder** **extends** RecyclerView**.**ViewHolder **{**

**static** UserViewHolder **create(**LayoutInflater inflater**,** ViewGroup parent**)** **{**

UserItemBinding binding **=** UserItemBinding

**.**inflate**(**inflater**,** parent**,** **false);**

**return** **new** **UserViewHolder(**binding**);**

**}**

**private** UserItemBinding mBinding**;**

**private** **UserViewHolder(**UserItemBinding binding**)** **{**

**super(**binding**.**getRoot**());**

mBinding **=** binding**;**

**}**

**public** **void** **bindTo(**User user**)** **{**

mBinding**.**setUser**(**user**);**

mBinding**.**executePendingBindings**();**

**}}**

要小心的一点细节就是调用这个executePendingBindings。这是必要的，因为当你的数据无效时，数据绑定实际上等到下一个动画帧设置布局之前。这不是我们可以批量处理数据中发生的所有更改并全部应用，而是因为RecyclerView并不喜欢它。RecyclerView调用BindView，它要你准备这个视图，以便它可以测量一个布局。这就是我们调用的原因executePendingBindings，以便数据绑定刷新所有待处理的更改。否则，它会创建另一个布局失效。你可能没有注意到它，但它将在操作列表中。

因为onCreateViewholder它只是调用第一个方法，onBind并将该对象传递给ViewHolder。而已。我们没有写任何findViewById，没有设置，纳达。一切都封装在您的布局文件中。

**public** UserViewHolder **onCreateViewHolder(**ViewGroup viewGroup**,** **int** i**)** **{**

**return** UserViewHolder**.**create**(**mLayoutInflater**,** viewGroup**);}public** **void** **onBindViewHolder(**UserViewHolder userViewHolder**,** **int** position**)** **{**

userViewHolder**.**bindTo**(**mUserList**.**get**(**position**));}**

在前面的代码中，我们展示了一个非常简单，直接的实现。比方说，你的用户对象的名字改变了。绑定系统将实现它，并在下一个动画框架中重新进行自我布局。下一个动画帧开始，计算已经改变的，并更新TextView。然后，TextView说：“好吧，我的文本已经改变了，我不得不重新布局布局，因为我不知道我的新尺寸。让我们去告诉RecyclerView，它的一个孩子是不高兴的，它也需要重新布局自己。“当这种情况发生时，你不会收到任何动画，因为你在发生一切事情之后告诉了RecyclerView 。RecyclerView将尝试修复自己，这将完成。****结果：没有动画****但是，这不是我们想要的。

我们想要发生的事情是当用户的对象失效时，我们告诉适配器该项目已经改变。反过来，它会告诉RecyclerView，“嘿，你的一个孩子将要改变，准备好自己”.RecyclerView将进行布局，对于那些孩子已经改变了的人来说，它将会指示他们重新绑定自己。当他们重新绑定时，TextView将会说：“好的，我的文本已经设置好了，我需要布局”.RecyclerView会说：“好的，别担心，我已经把你放了出来，让我来衡量你。” ****结果：多姿的动画****。你会得到所有的动画，因为一切都发生在RecyclerView的控制之下。

### Rebind Callback和 Payload[（25:50）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,76,true);)

这实际上是我们需要作为图书馆发行的一部分，但在此期间，我想让你知道如何做到这一点。在数据绑定中，我们有这个API可以添加rebind回调。这基本上是一个回调，你可以附加，然后得到通知时，数据绑定即将计算。例如，也许你可能想要冻结UI的变化。你可以勾选这个onPreBind，在这一点你可以返回一个布尔值，你可以说：“不，不要再重新绑定”。如果其中一个听众说的话，数据绑定会打电话给：“嘿，我取消了重新绑定。现在有责任给我打电话，因为我什么都不做。“

现在我们要做的就是如果RecyclerView不计算你的布局，返回false。查看，当RecyclerView没有做你的计算时，你不应该更新自己。这就是计算布局，今年夏天发布的新的RecyclerView API。而当onCanceled我们来到时，我们只是告诉适配器，“嘿，这个东西已经改变了，找出来”，因为我们已经知道他们在持有者的位置。然后，让RecyclerView决定要做什么。

**public** UserViewHolder **onCreateViewHolder(**ViewGroup viewGroup**,** **int** i**)** **{**

**final** UserViewHolder holder **=** UserViewHolder**.**create**(**mLayoutInflater**,**

viewGroup**);**

holder**.**getBinding**().**addOnRebindCallback**(new** OnRebindCallback**()** **{**

**public** **boolean** **onPreBind(**ViewDataBinding binding**)** **{**

**return** mRecyclerView **!=** **null** **&&** mRecyclerView**.**isComputingLayout**();**

**}**

**public** **void** **onCanceled(**ViewDataBinding binding**)** **{**

**if** **(**mRecyclerView **==** **null** **||** mRecyclerView**.**isComputingLayout**())** **{**

**return;**

**}**

**int** position **=** holder**.**getAdapterPosition**();**

**if** **(**position **!=** RecyclerView**.**NO\_POSITION**)** **{**

notifyItemChanged**(**position**,** DATA\_INVALIDATION**);**

**}**

**}**

**});**

**return** holder**;}**

以前，我们只有一个onBind方法，所以我们开始编写这个新的RecyclerView API，在那里你得到一个有效载荷的列表。这是在该ViewHolder上更改的事物列表。关于这个API很酷的事情是，只有当RecyclerView重新绑定到相同的视图时，才会收到有效载荷。你知道这个视图已经代表了同一个项目，但是你想要执行的只是一些改变（希望是语法上的改变）。我们发送的数据无效负载回到这里。如果因为数据绑定而来，我们只是打电话executePendingBindings。你还记得我们没有让它自己更新吗？现在是时候更新自己，因为RecyclerView告诉它。

如果您想知道这是什么样子，数据绑定只是遍历有效载荷，并检查是否这个数据验证是唯一的有效载荷。例如，也许别人正在发送你不知道的有效载荷，你应该保释出来，因为你不知道这些改变是什么。

**public** **void** **onBindViewHolder(**UserViewHolder userViewHolder**,** **int** position**)** **{**

userViewHolder**.**bindTo**(**mUserList**.**get**(**position**));}public** **void** **onBindViewHolder(**UserViewHolder holder**,** **int** position**,**

List**<**Object**>** payloads**)** **{**

notifyItemChanged**(**position**,** DATA\_INVALIDATION**);...}**

我们将把它作为一个图书馆出货，因为它给了你表演，给你动画，使一切更美好，并使RecyclerView开心。数据绑定大多是一个快乐的孩子！

数据失效只是一个简单的对象，但我想显示它，以防万一你好奇：

**static** Object DATA\_INVALIDATION **=** **new** Object**();private** **boolean** **isForDataBinding(**List**<**Object**>** payloads**)** **{**

**if** **(**payloads **==** **null** **||** payloads**.**size**()** **==** 0**)** **{**

**return** **false;**

**}**

**for** **(**Object obj **:** payloads**)** **{**

**if** **(**obj **!=** DATA\_INVALIDATION**)** **{**

**return** **false;**

**}**

**}return** **true;}**

### 多种视图类型[（28:50）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,81,true);)

数据绑定的另一个用例是多视图类型。这总是发生：你有一个标题视图，或者你有一个应用程序显示谷歌搜索结果，你可以有一个照片结果或地方的结果。你怎么能在RecyclerView中构造这个？假设您有一个使用变量的布局文件，您将其命名为“数据”。这个名称“数据”非常重要，因为您要重复使用相同的名称。您使用常规布局文件：

<layout>

<data>

<variable name="data" type="com.example.Photo"/>

</data>

<ImageView android:src="@{data.url}" …/></layout>

如果你需要另一种类型的结果，例如叫做“place”的东西，那么你需要一个完全不同的布局，另一个XML文件：

<layout>

<data>

<variable name="data" type="com.example.Place"/>

</data>

<ImageView android:src="@{data.url}" …/></layout>

这两个布局文件之间唯一共享的就是变量名称，称为“数据”。当我们这样做时，我们创建了一个名为dataBoundViewHolder。

**public** **class** **DataBoundViewHolder** **extends** RecyclerView**.**ViewHolder **{**

**private** ViewDataBinding mBinding**;**

**public** **DataBoundViewHolder(**ViewDataBinding binding**)** **{**

**super(**binding**.**getRoot**());**

mBinding **=** binding**;**

**}**

**public** ViewDataBinding **getBinding()** **{**

**return** mBinding**;**

**}**

**public** **void** **bindTo(** Place place**)** **{**

mBinding**.**setPlace**(**place**);**

mBinding**.**executePendingBindings**();**

**}}**

这和前面的例子是一样的。这是一个真正的数据绑定对象，保持绑定。真正的数据绑定是所有生成的类的基类。这就是为什么你通常可以保持参考。我们创建这个绑定方法 - 以前，它绑定到用户，现在是放置。

不幸的是，这里有一个问题。setPlaceReal Data Binding类中没有方法，因为它是基类。相反，基类提供了另一个API，基本上是setVariable：

**public** **void** **bindTo(** Object obj**)** **{**

mBinding**.**setVariable**(**BR**.**data**,** obj**);**

mBinding**.**executePendingBindings**();}**

您可以提供变量的标识符，然后提供您想要的任何对象，如常规的Java对象。生成的类将检查类型将分配它。

set变量看起来像这样，它基本上说：“如果过去的ID是我知道的ID之一，把它投下来，并分配它。

**boolean** **setVariable(int** id**,** Object obj**)** **{**

**if** **(**id **==** BR**.**data**)** **{**

setPhoto**((**Photo**)** obj**);**

**return** **true;**

**}**

**return** **false;}**

一旦你这样做onBind，onCreate方法是完全一样的。我们所做的是getItemViewType，所以在视图类型中，我们返回布局ID作为类型的ID。这很好，因为当我们返回布局ID和获取项目离开类型时，RecyclerView将它传递回来onCreateViewHolder，它将通过DataBindingUtil来创建正确的绑定类。每个项目都有自己的布局，你不必布局一个对象。

DataBoundViewHolder **onCreateViewHolder(**ViewGroup viewGroup**,** **int** type**)** **{**

**return** DataBoundViewHolder**.**create**(**mLayoutInflater**,** viewGroup**,** type**);}void** **onBindViewHolder(**DataBoundViewHolder viewHolder**,** **int** position**)** **{**

viewHolder**.**bindTo**(**mDataList**.**get**(**position**));}public** **int** **getItemViewType(int** position**)** **{**

Object item **=** mItems**.**get**(**position**);**

**if** **(**item **instanceof** Place**)** **{**

**return** R**.**layout**.**place\_layout**;**

**}** **else** **if** **(**item **instanceof** Photo**)** **{**

**return** R**.**layout**.**photo\_layout**;**

**}**

**throw** **new** **RuntimeException(**"invalid obj"**);}**

当然，如果你是在生产应用程序中写这个，你可能会保留做实例检查。你应该有一个知道如何返回布局的基类，但是你得到了一般的想法。

### Binding Adapters和Callbacks[（31:27）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,91,true);)

根据大众民意调查（我所做的），为数据绑定中最酷的功能做好准备。它甚至可能是所有Android中最酷的功能。好的，也许我有点夸大其词。

假设你有一些比setText图像网址更复杂的东西。你想设置为ImageView，并且你想要设置一个图片的URL。当然，你不想在UI线程上这样做（记住，这些东西是在UI线程上评估的）。你想使用毕加索或其他图书馆之一。也许你会做出这样的表达？

<ImageView …

android:src="@{contact.largeImageUrl}" />

这不是很有效。这个背景从哪里来，你把它放到什么地方？没有意见。如果你写这个，你会失去工作。相反，我们要做的是创建一个BindingAdapter。

**@BindingAdapter(**"android:src"**)public** **static** **void** **setImageUrl(**ImageView view**,** String url**)** **{**

Picasso**.**with**(**view**.**getContext**()).**load**(**url**).**into**(**view**);}**

现在BindingAdapter是一个注释。这是Android的源代码，因为我们正在设置属性android：src。这是一个公共的静态方法，它需要两个参数。它需要一个视图和一个字符串，但是请注意，它也可以采用其他类型。例如，如果你想要一个不同的int或drawable，你也可以这样做。然后你把它填满，你可以把任何你想要的在这里。在这种情况下，我们已经放入了毕加索的东西。我们所有的代码都在那里。现在我们有了这个观点，我们可以得到上下文。我们可以做任何我们想要的代码。我们现在可以加载UI线程，就像我们想要的一样。

### 和View属性一起使用[（33:12）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,97,true);)

您也可能想要做更复杂的事情，例如在这种情况下，我们有PlaceHolder，源和图像URL。

<ImageView …

android:src="@{contact.largeImageUrl}"

app:placeHolder="@{R.drawable.contact\_placeholder}"/>

我们有两个不同的属性，他们有两个不同的静态方法，所以这是行不通的。实际上，现在我们可以在同一个BindingAdapter中拥有两个属性。您只需传递两个值，然后在中间填入您的毕加索编码。

<ImageView …

android:src="@{contact.largeImageUrl}"

app:placeHolder="@{R.drawable.contact\_placeholder}"/>

**@BindingAdapter(**value **=** **{**"android:src"**,** "placeHolder"**},**

requireAll **=** **false)public** **static** **void** **setImageUrl(**ImageView view**,** String url**,**

**int** placeHolder**)** **{**

RequestCreator requestCreator **=**

Picasso**.**with**(**view**.**getContext**()).**load**(**url**);**

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

requestCreator**.**placeholder**(**placeHolder**);**

**}**

requestCreator**.**into**(**view**);}**

如果你现在有三个呢？您必须拥有一个Android来源，Android PlaceHolder和Android图片网址。所有这些不同的BindingAdapter。真的，我有点太懒了，所以让我们做点别的。我们可以有一个BindingAdapter，它可以把所有这些，甚至只有一个或两个，或它们的任意组合。我们所要做的就是将所需的全部设置为假，并且我们将所有这些参数。这将通过所有这些价值观。如果没有提供，它会通过它们作为默认值。如果在布局中没有PlaceHolder属性，则PlaceHolder将为零。我们在那之前，我们检查那毕加索的二传手。

### Previous Values[（34:55）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,98,true);)

你有时也需要以前的值。在这个例子中，我们有一个OnLayoutChanged。

**@BindingAdapter(**"android:onLayoutChange"**)public** **static** **void** **setOnLayoutChangeListener(**View view**,**

View**.**OnLayoutChangeListener oldValue**,**

View**.**OnLayoutChangeListener newValue**)** **{**

**if** **(**Build**.**VERSION**.**SDK\_INT **>=** Build**.**VERSION\_CODES**.**HONEYCOMB**)** **{**

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

view**.**removeOnLayoutChangeListener**(**oldValue**);**

**}**

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

view**.**addOnLayoutChangeListener**(**newValue**);**

**}**

**}}**

在添加新的之前，我们要删除旧的，但在这种情况下，我们不知道旧的是什么。我们可以添加新的，这很容易，但我们如何删除旧的？那么，你也可以拿这个价值，我们会把它给你。如果你有这样的代码，我们会为你保留原来的价值。所以，如果你有一些巨大的东西，那实际上只是短暂的，我们仍然会坚持下去。然而，对于这样的情况，无论如何它将在你的记忆中，这是很好的。每次它改变，要开始正确的动画，你想知道什么是以前。

只要使用这个API，数据绑定就能为您做出想法。你只需要考虑如何使变化生动起来。当然，你也可以用多个属性来做到这一点。我们将要通过你。你所要做的就是把所有旧的价值，然后所有的新价值。

### 隐性注入[（36:20）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,99,true);)

假设我们有这个适配器：

**public** **interface** **TestableAdapter** **{**

**@BindingAdapter(**"android:src"**)**

**void** **setImageUrl(**ImageView imageView**,** String url**);}public** **interface** **DataBindingComponent** **{**

TestableAdapter **getTestableAdapter();}**DataBindingUtil**.**setDefaultComponent**(**myComponent**);**

‐ or ‐binding **=** MyLayoutBinding**.**inflate**(**layoutInflater**,** myComponent**);**

显然会发生什么事情是当我们的绑定代码调用时，它将称之为我的活页夹setImageUrl。如果我有一些状态，我想在我的BindingAdapter？或者，假设我有不同种类的BindingAdapters，这取决于我在应用程序中所做的事情。在这种情况下，这将是一个痛苦。我们真正想要的只是BindingAdapter的一个实例。这是从哪里来的？

我们可以做的是创建一个绑定组件DataBindingComponent，它是一个接口。当你有一个实例方法时，我们将生成这个让我的适配器进入这个接口。然后，由您来实现它。我们不知道如何实现它，但是实现它，然后只设置默认组件。

你也可以在每个布局的基础上做到这一点。在这种情况下，您可以设置默认值，并可以在所有布局中使用。然后我们确切知道使用什么组件来获取适配器。

您可能还想使用您的组件作为参数。例如，我们刚才看到了这个setImageURL。

**@BindingAdapter(**"android:src"**)public** **static** **void** **setImageUrl(**MyAppComponent component**,**

ImageView view**,**

String imageUrl**)** **{**

component**.**getImageCache**().**loadInto**(**view**,** imageUrl**);}**

我们想要使用某种状态。我们假设这就是图像缓存，我们想要使用该图像缓存加载图像。总和状态从哪里来？我们要做的就是使用这个组件。你可以放任何你想要的方法：在这种情况下，它是[somestate] .get [somestate]。你将把它作为你的BindingAdapter的第一个参数传入，然后你可以用它做任何你想做的事情。我们不知道你在用你的组件做什么，对吧？无论你想做什么，所以它可以非常方便。

### 事件处理程序[（38:56）](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,101,true);)

我们有这个onClick属性，我们有一个clicked处理程序的方法。clicked可能是getClicked，或者isClicked，或者它可能是一个领域，“点击”，所以我们怎么知道在这种情况下做什么？

<Button …

android:onClick="@{isAdmin ? handler.adminClick : handler.userClick}" />

*// No "setOnClick" method for View. Need a way to ﬁnd it.***@BindingMethods({**

**@BindingMethod(**type **=** View**.**class**,**

attribute **=** "android:onClick"**,**

method **=** "setOnClickListener"**})** *// Look for setOnClickListener in View* **void** **setOnClickListener(**View**.**OnClickListener l**)** *// Look for single abstract method in OnClickListener* **void** **onClick(**View v**);**

首先，我们需要找出什么onClick手段？我们知道onClick不是setOnClick，因为我们看，我们看到没有setOnClick，但有这个约束的方法。它说的onClick意思setOnClickListener。所以我们来看看setOnClickListener，它有一个参数：它需要一个onClickListener参数，所以让我们来看看。

我们看到只有一个抽象方法onClickListener，所以我们知道这可能是您想要用于事件处理程序的侦听器。现在我们看看处理程序，并在那里找到一个叫做的方法clicked。

**static** **class** **OnClickListenerImpl1** **implements** OnClickListener **{**

**public** Handler mHandler**;**

**@Override**

**public** **void** **onClick(**android**.**view**.**View arg0**)** **{**

mHandler**.**adminClick**(**arg0**);**

**}}static** **class** **OnClickListenerImpl2** **implements** OnClickListener **{**

**public** Handler mHandler**;**

**@Override**

**public** **void** **onClick(**android**.**view**.**View arg0**)** **{**

mHandler**.**userClick**(**arg0**);**

**}}**

我们找到了这个clicked方法，它采用了相同的参数。我们有一场比赛，耶！我们知道这是一个事件处理程序，所以我们知道该怎么做：我们会把它当作一个事件来对待。

亦酒读读 因为没有单一的抽象方法。那里有三个。在这种情况下，你所做的是你自己组成接口，然后把它们合并在一起。这基本上就是我所做的，我将它们合并在一起。基本上你正在做的是你正在合并之前和之后的所有变化。如果他们是空的，那么你什么都不做，如果他们不是空的，那么你做一些事情。所需的全部当然是必需的。

# 英文版本：

Get it straight from the horse[Data Binding](https://developer.android.com/tools/data-binding/guide.html)in Android. Step two: profit [Bay Area Android Dev Group](http://www.meetup.com/bayareaandroid/), they demonstrate how using Data Bindings can improve your application by removing boilerplate for data-driven UI, allowing you to write cleaner, better code.

### Introduction[(0:00)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,0,true);)

We are George Mount and Yigit Boyar, and we work on the Android UI Toolkit team. We have a lot of information about Data Binding to share with you, and lots of code to go with it. We

### Why Data Binding?[(0:44)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,1,true);)

You may wonder why we decided to implement this library. Here

<LinearLayout >

<TextView android:id="@+id/name"/>

<TextView android:id="@+id/lastName"/></LinearLayout>

This is an Android UI you see all the time. Say you have a bunch of videos with IDs. Your designer comes and says,

**private** TextView mName**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

mName **=** **(**TextView**)** findViewById**(**R**.**id**.**name**);}**

**public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

mName**.**setText**(null);**

**}** **else** **{**

mName**.**setText**(**user**.**getName**());**

**}}**

You write a new TextView, you find it from the UI, and you set your logic so that whenever you need to update your user, you have to set the information on the TextView.

**private** TextView mName**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

mName **=** **(**TextView**)** findViewById**(**R**.**id**.**name**);**

mLastName **=** **(**TextView**)** findViewById**(**R**.**id**.**lastName**);}public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

mName**.**setText**(null);**

mLastName**.**setText**(null);**

**}** **else** **{**

mName**.**setText**(**user**.**getName**());**

mLastName**.**setText**(**user**.**getLastName**());**

**}}**

All in all, that is a lot of things you have to do just to add one view to your UI. It seems like too much stupid boilerplate code that doesn

There are already some really nice libraries to make this easier and more solid. For example, if you use ButterKnife, you could get two of those ugly viewByIds, making it much easier to read. You can get rid of the extra code, telling ButterKnife to delete it for you.

**private** TextView mName**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

ButterKnife**.**bind**(this);}public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

mName**.**setText**(null);**

mLastName**.**setText**(null);**

**}** **else** **{**

mName**.**setText**(**user**.**getName**());**

mLastName**.**setText**(**user**.**getLastName**());**

**}}**

It

**private** Holdr\_ActivityMain holder**;protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

setContentView**(**R**.**layout**.**activity\_main**);**

holder **=** **new** Holdr\_ActivityMain**(**findViewById**(**content**));}public** **void** **updateUI(**User user**)** **{**

**if** **(**user **==** **null)** **{**

holder**.**name**.**setText**(null);**

holder**.**lastName**.**setText**(null);**

**}** **else** **{**

holder**.**name**.**setText**(**user**.**getName**());**

holder**.**lastName**.**setText**(**user**.**getLastName**());**

**}}**

This is better again, but there

When you use Data Binding, it

**private** ActivityMainBinding mBinding**;protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

mBinding **=** DataBindingUtil**.**setContentView**(this,**

R**.**layout**.**activity\_main**);}**

**public** **void** **updateUI(**User user**)** **{**

mBinding**.**setUser**(**user**);}**

### Behind the Scenes[(3:53)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,15,true);)

How does Data Binding work behind the scenes? Take a look at the layout file from before:

<LinearLayout >

<TextView android:id="@id/name" />

<TextView android:id="@id/lastName" /></LinearLayout>

I have these IDs, but why do I need them if I could find them back in my Java code? I actually don

<LinearLayout >

<TextView android:text="@{user.name}"/>

<TextView android:text="@{user.lastName}"/></LinearLayout>

Now, when I look at this layout file, I know what the TextView shows. It has become very obvious, so I don

<layout>

<data>

<variable name="user"

type="com.android.example.User"/>

</data>

<LinearLayout >

<TextView android:text="@{user.name}"/>

<TextView android:text="@{user.lastName}"/>

<TextView android:text='@{"" + user.age}'/>

</LinearLayout></layout>

It

### But How Does It Work?[(5:57)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,19,true);)

The first thing the Data Binding library does is process your layout files. By

Get more development news like this

窗体顶端

Subscribe



Comments

窗体底端

The second step is to parse these expressions by running it through a grammar. For example, in this case:

<TextView android:visibility="@user.isAdmin ? View.VISIBLE : View.GONE}"/>

Theuseris an ID, theViewis an ID, and the otherViewis an ID. They

The third step is resolving dependencies, which happens when your code is being compiled. In this step, for example, we look atuser.isAdminand figure out what it means. We think

The final step is writing data binders. We write the classes that YOU don

### An Example Case[(7:40)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,27,true);)

Here is an actual case of a layout file.

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

<data>

<variable name="user" type="com.android.example.User"/>

</data>

<RelativeLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView android:text="@{user.name}"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<TextView android:text="@{user.lastname}"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

</RelativeLayout></layout>

As we process, we get rid of everything the view system doesn

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView android:tag="binding\_1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<TextView android:tag="binding\_2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

</RelativeLayout>

This is actually how we make Data Binding backwards compatible. When you put it on a Gingerbread device, the poor guy has no idea what

### Expression Tree[(8:01)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,30,true);)

<TextView android:text="@{user.age < 18 ? @string/redacted : user.name}"/>

Here

TheresetText(CharSequence). Now, Data Binding knows how to turn that expression into Java code. If you go into detail, there

<TextView android:text="@{myVariable}"/>

textView.setText(myVariable);<ImageView android:src="@{user.image}"/>

imageView.setSrc(user.image);

ImageView is a source attribute, so would it be correct, as in the above example, to usesetSrc? No, because there

It

<TextView />

textView.setText(myVariable);<ImageView android:src="@{user.image}"/>

imageView.setImageResource(user.image);

@BindingMethod(

type = android.widget.ImageView.class,

attribute = "android:src",

method = "setImageResource")

We have these annotations that we create, where you can simply say,

### Data Binding Goodies[(9:54)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,36,true);)

Data Binding makes your life a lot easier. LetmostlyJava. It allows things like field access, method calls, parameters, addition, comparisons, index access on arrays, constant access, and even ternary expressions. Thatnew. We really donnewin your expressions.

Our basic goal is to make this thing as****short and readable****as possible in your expressions, right in your XML. We doncontact.name. We look at it and think

We also do automatic null checks, which is actually really, really cool. If you want to access the name, but contact is null, how much of pain in the neck would it be to writecontact null ? null : contact.friend null ? :? You don

We also have the null coalescing operator, which you may have seen from other languages. It

contact**.**lastName **??** contact**.**namecontact**.**lastName **!=** **null** **?** contact**.**lastName **:** contact**.**name

It says if the first one is not null, choose the first one. If it is null, then choose the second one.

We also have list access and map access using the bracket operator. If you havecontacts[0], that contact could be a list or an array, it

### Resources[(12:20)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,48,true);)

We want you to be able use resources in your expressions. What would an expression language be in Android without resources? Now you can use resources and string formatting right in your expressions.

In Expressions:

**android:**padding**=**"@{isBig ? @dimen/bigPadding : @dimen/smallPadding}"

Inline string formatting:

**android:**text**=**"@{@string/nameFormat(firstName, lastName)}"

Inline plurals:

**android:**text**=**"@{@plurals/banana(bananaCount)}"

### Automagic Attributes[(13:00)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,49,true);)

Here we have a DrawerLayout

<android.support.v4.widget.DrawerLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

app:scrimColor="@{@color/scrim}"/>

drawerLayout**.**setScrimColor**(**

resources**.**getColor**(**R**.**color**.**scrim**))**

We have this attributeapp:scrimColor. TheresetScrimColor. We look for thissetScrimColorwhen we have an attribute with a namescrimColor, and we check if the types match. First we look at color, which is anint. IfsetScrimColortakes anint, it

### Event Handlers[(13:41)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,50,true);)

I donclickedusing a button or a view, but we also support it here in Data Binding. You can use aclicked, but now any of the events are supported as well. Of course, this works back to Gingerbread. You can even do things where you have assigned an arbitrary event handler as part of an expression (IonTextChanged. TextWatcher has three methods on it, but everybody only cares aboutonTextChanged, right? You can actually access just one of them if you want, or all of them.

<Button android:onClick="clicked" />

<Button android:onClick="@{handlers.clicked}" />

<Button android:onClick="@{isAdult ? handlers.adultClick : handlers.childClick}" />

<Button android:onTextChanged="@{handlers.textChanged}" />

### Observability in Detail[(14:56)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,51,true);)

What happens when you update your view? Imagine we have a store, and we have an item whose price has recently changed. This has has to automatically update our UI. How does that happen? With Data Binding, that happens really cheaply and easily.

The first thing we have to do is create an item, some kind of observable object. Here, I

**public** **class** **Item** **extends** BaseObservable **{**

**private** String price**;**

**@Bindable**

**public** String **getPrice()** **{**

**return** **this.**name**;**

**}**

**public** **void** **setPrice(**String price**)** **{**

**this.**price **=** price**;**

notifyPropertyChanged**(**BR**.**price**);**

**}}**

We notify it by adding in thisnotifyPropertyChanged. But what do we notify that@Bindableannotation on thegetPrice. That generates thisBR.price, the price field in the BR class. The BR is like the R class, we just generate it for you and it just sucks in these binding resources. However, you may not want us to invade your whole hierarchy, so we allow you to implement the observable class as well. Yes, I hear the has-a vs. is-a people complaining

**public** **class** **Item** **implements** Observable **{**

**private** PropertyChangeRegistry callbacks **=** **new** **@Override**

**public** **void** **addOnPropertyChangedCallback(**

OnPropertyChangedCallback callback**)** **{**

callbacks**.**add**(**callback**);**

**}**

**@Override**

**public** **void** **removeOnPropertyChangedCallback(**

OnPropertyChangedCallback callback**)** **{**

callbacks**.**remove**(**callback**);**

**}}**

We have this convenient class called thePropertyChangedRegistrythat lets you essentially take those callbacks and notify them. Some of you might think this is just a pain in the neck, and instead want to have an observable field. Essentially, each of these is an observable object, and it just has one entry in it. It

The special thing about these objects is that in the Java code, you have to call the set and get methods, but in your binding expressions, you can just sayitem.price, and we will know that we need to call the getter. So when the price changes, it just sets the value.

**public** **class** **Item** **{**

**public** **final** ObservableField**<**Drawable**>** image **=**

**new** ObservableField**<>();**

**public** **final** ObservableField**<**String**>** price **=**

**new** ObservableField**<>();**

**public** **final** ObservableInt inventory **=**

**new** **ObservableInt();}**

item**.**price**.**set**(**"$33.41"**);**

In other cases you may have more

ObservableMap**<**String**,** Object**>** item **=**

**new** ObservableArrayMap**<>();**

item**.**put**(**"price"**,** "$33.41"**);**

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text='@{item["price"]}'/>

### Notify on Any Thread[(18:29)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,62,true);)

One of the conveniences here is that you don

### Performance[(19:21)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,63,true);)

Perhaps the most important consideration for this project was to not make it slow. Data Binding is infamous for being slow, so in Android, we were double careful to take that into consideration, and we believe we did a good job.

The foremost aspect of performance is that there is basically zero reflection. Everything happens in compile time. Occasionally, things are inconvenient because it happens in the compile time, but in the long run we don

The second part is something nice that you get for free. Let

Another performance benefit in Data Binding comes in cases where you have two expressions such as these:

<TextView android:text="@{user.address.street}"/><TextView android:text="@{user.address.city}"/>

You haveuser.addressand anotheruser.address. The code DataBinding will generate looks like this:

Address address **=** user**.**getAddress**();**String street **=** address**.**getStreet**();**String city **=** address**.**getCity**();**

It

Another positive side effect of the performance is thefindById. When you codefindByIdon the view on Android, it actually goes to all of its children, and says something like findViewByIda second time for the other view, and the same thing happens again.

However, when you initialize Data Binding, we actually know which views we are interested in at compile time, so we have a method of finding all the views we want. We traverse the layout hierarchy once to collect all the views. It

Performance is about the little details. You

### RecyclerView and Data Binding[(22:14)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,71,true);)

Using ViewHolders was very common for ListViews, and RecyclerView enforces this pattern. If you look at what Data Binding generates, you

**public** **class** **UserViewHolder** **extends** RecyclerView**.**ViewHolder **{**

**static** UserViewHolder **create(**LayoutInflater inflater**,** ViewGroup parent**)** **{**

UserItemBinding binding **=** UserItemBinding

**.**inflate**(**inflater**,** parent**,** **false);**

**return** **new** **UserViewHolder(**binding**);**

**}**

**private** UserItemBinding mBinding**;**

**private** **UserViewHolder(**UserItemBinding binding**)** **{**

**super(**binding**.**getRoot**());**

mBinding **=** binding**;**

**}**

**public** **void** **bindTo(**User user**)** **{**

mBinding**.**setUser**(**user**);**

mBinding**.**executePendingBindings**();**

**}}**

One little detail to be careful about is to call thisexecutePendingBindings. This is necessary because when your data invalidates, Data Binding actually waits until the next animation frame before it sets the layout. This is not so that we can batch all the changes that happen in your data and apply all it once, but because RecyclerView doesnexecutePendingBindings, so that Data Binding flushes all pending changes. Otherwise, it

ForonCreateViewholder, it simply calls the first method, andonBindpasses the object to the ViewHolder. ThatfindViewById, no settings, nada. Everything is encapsuled in your layout file.

**public** UserViewHolder **onCreateViewHolder(**ViewGroup viewGroup**,** **int** i**)** **{**

**return** UserViewHolder**.**create**(**mLayoutInflater**,** viewGroup**);}**

**public** **void** **onBindViewHolder(**UserViewHolder userViewHolder**,** **int** position**)** **{**

userViewHolder**.**bindTo**(**mUserList**.**get**(**position**));}**

In the previous code, we showed a very simple, straightforward implementation. Say, for instance, that your user objectaftereverything happened. RecyclerView will try to fix itself, it will be done.****Result: NO ANIMATIONS****But that

What we wanted to happen was that when the user

****Result: MUCHO ANIMATIONS****. You will get all the animations because everything happened under the control of RecyclerView.

### Rebind Callback and Payload[(25:50)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,76,true);)

This is actually the part we need to release as a library, but in the meantime, I want to let you know how you can do this. In Data Binding, we have this API where you can add a rebind callback. ItonPreBind, at which point you get to return a boolean where you can say,

Now all we have to do here is if RecyclerView is not calculating your layout, return false. View, you should not update yourself when RecyclerView is not doing your computation. That is computing layouts, the new RecyclerView API that was released this summer. And when theonCanceledcomes, we just tell the adapter that,

**public** UserViewHolder **onCreateViewHolder(**ViewGroup viewGroup**,** **int** i**)** **{**

**final** UserViewHolder holder **=** UserViewHolder**.**create**(**mLayoutInflater**,**

viewGroup**);**

holder**.**getBinding**().**addOnRebindCallback**(new** OnRebindCallback**()** **{**

**public** **boolean** **onPreBind(**ViewDataBinding binding**)** **{**

**return** mRecyclerView **!=** **null** **&&** mRecyclerView**.**isComputingLayout**();**

**}**

**public** **void** **onCanceled(**ViewDataBinding binding**)** **{**

**if** **(**mRecyclerView **==** **null** **||** mRecyclerView**.**isComputingLayout**())** **{**

**return;**

**}**

**int** position **=** holder**.**getAdapterPosition**();**

**if** **(**position **!=** RecyclerView**.**NO\_POSITION**)** **{**

notifyItemChanged**(**position**,** DATA\_INVALIDATION**);**

**}**

**}**

**});**

**return** holder**;}**

Previously, we only had the oneonBindmethod, so we started writing this new RecyclerView API, where you get a list of payloads. ItexecutePendingBindings. Do you remember we didn

If you

**public** **void** **onBindViewHolder(**UserViewHolder userViewHolder**,** **int** position**)** **{**

userViewHolder**.**bindTo**(**mUserList**.**get**(**position**));}**

**public** **void** **onBindViewHolder(**UserViewHolder holder**,** **int** position**,**

List**<**Object**>** payloads**)** **{**

notifyItemChanged**(**position**,** DATA\_INVALIDATION**);...}**

We will ship this as a library, because it gives you performance, it gives you animations, makes everything nicer, and makes RecyclerView happy. Data Binding is mostly a happy child!

Data Invalidation is just a simple object, but I want to show it in case you

**static** Object DATA\_INVALIDATION **=** **new** Object**();private** **boolean** **isForDataBinding(**List**<**Object**>** payloads**)** **{**

**if** **(**payloads **==** **null** **||** payloads**.**size**()** **==** 0**)** **{**

**return** **false;**

**}**

**for** **(**Object obj **:** payloads**)** **{**

**if** **(**obj **!=** DATA\_INVALIDATION**)** **{**

**return** **false;**

**}**

**}return** **true;}**

### Multiple View Types[(28:50)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,81,true);)

Another use case of Data Binding is multiple view types. This always happens: you have a header view, or maybe you have an application which shows search results from Google, where you can have a photo result or a place result. How can you structure this in RecyclerView? Let

<layout>

<data>

<variable name="data" type="com.example.Photo"/>

</data>

<ImageView android:src="@{data.url}" /></layout>

If you need another type of result, for example something called

<layout>

<data>

<variable name="data" type="com.example.Place"/>

</data>

<ImageView android:src="@{data.url}" /></layout>

The only thing shared between these two layout files is the variable name, which is called dataBoundViewHolder.

**public** **class** **DataBoundViewHolder** **extends** RecyclerView**.**ViewHolder **{**

**private** ViewDataBinding mBinding**;**

**public** **DataBoundViewHolder(**ViewDataBinding binding**)** **{**

**super(**binding**.**getRoot**());**

mBinding **=** binding**;**

**}**

**public** ViewDataBinding **getBinding()** **{**

**return** mBinding**;**

**}**

**public** **void** **bindTo(** Place place**)** **{**

mBinding**.**setPlace**(**place**);**

mBinding**.**executePendingBindings**();**

**}}**

This is the same implementation as the previous example. It is a Real Data Binding object that keeps the binding. Real Data Binding is a base class for all generated classes. This is why you can usually keep the reference. We create this bind method

Unfortunately, theresetPlacemethod in the Real Data Binding class, because itsetVariable:

**public** **void** **bindTo(** Object obj**)** **{**

mBinding**.**setVariable**(**BR**.**data**,** obj**);**

mBinding**.**executePendingBindings**();}**

You can provide the identifier of the variable, and then whatever object you want, like a regular Java object. The generated class is going to check the type will assign it.

The set variable looks something like this, which basically says

**boolean** **setVariable(int** id**,** Object obj**)** **{**

**if** **(**id **==** BR**.**data**)** **{**

setPhoto**((**Photo**)** obj**);**

**return** **true;**

**}**

**return** **false;}**

Once you do this, theonBind,onCreatemethods are exactly the same. What we do isgetItemViewType, so the in the view type, we return the layout ID as the ID of the type. This works very well because when we return the layout ID and the get item leave type, RecyclerView passes it back onto theonCreateViewHolder, which will pass through the DataBindingUtil to create the correct binding class for that. Every item has its own layout, you don

DataBoundViewHolder **onCreateViewHolder(**ViewGroup viewGroup**,** **int** type**)** **{**

**return** DataBoundViewHolder**.**create**(**mLayoutInflater**,** viewGroup**,** type**);}**

**void** **onBindViewHolder(**DataBoundViewHolder viewHolder**,** **int** position**)** **{**

viewHolder**.**bindTo**(**mDataList**.**get**(**position**));}**

**public** **int** **getItemViewType(int** position**)** **{**

Object item **=** mItems**.**get**(**position**);**

**if** **(**item **instanceof** Place**)** **{**

**return** R**.**layout**.**place\_layout**;**

**}** **else** **if** **(**item **instanceof** Photo**)** **{**

**return** R**.**layout**.**photo\_layout**;**

**}**

**throw** **new** **RuntimeException(**"invalid obj"**);}**

Of course, if you were writing this in a production application, you would probably reserve doing instance check. You should probably have a base class that knows how to return the layout, but you get the general idea.

### Binding Adapters and Callbacks[(31:27)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,91,true);)

Prepare yourselves for the coolest feature in Data Binding, according to popular polls

LetsetText, for example an image URL. You want to set as ImageView, and you want to set an image URL. Of course, you don

<ImageView android:src="@{contact.largeImageUrl}" />

That

**@BindingAdapter(**"android:src"**)public** **static** **void** **setImageUrl(**ImageView view**,** String url**)** **{**

Picasso**.**with**(**view**.**getContext**()).**load**(**url**).**into**(**view**);}**

Now the BindingAdapter here is an annotation. This one

### Attributes Working Together[(33:12)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,97,true);)

You also might want to do something even more complex, for example in this case where we have the PlaceHolder, the source, and the image URL.

<ImageView android:src="@{contact.largeImageUrl}"

app:placeHolder="@{R.drawable.contact\_placeholder}"/>

We have two different attributes, and they have two different static methods, so that

<ImageView android:src="@{contact.largeImageUrl}"

app:placeHolder="@{R.drawable.contact\_placeholder}"/>

**@BindingAdapter(**value **=** **{**"android:src"**,** "placeHolder"**},**

requireAll **=** **false)public** **static** **void** **setImageUrl(**ImageView view**,** String url**,**

**int** placeHolder**)** **{**

RequestCreator requestCreator **=**

Picasso**.**with**(**view**.**getContext**()).**load**(**url**);**

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

requestCreator**.**placeholder**(**placeHolder**);**

**}**

requestCreator**.**into**(**view**);}**

What if you have three now? You have to have one Android source, Android PlaceHolder, and Android image URL. All these different BindingAdapters. Really, I am a little too lazy for that, so let

### Previous Values[(34:55)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,98,true);)

You also sometimes need previous values. In this example, we have an OnLayoutChanged.

**@BindingAdapter(**"android:onLayoutChange"**)public** **static** **void** **setOnLayoutChangeListener(**View view**,**

View**.**OnLayoutChangeListener oldValue**,**

View**.**OnLayoutChangeListener newValue**)** **{**

**if** **(**Build**.**VERSION**.**SDK\_INT **>=** Build**.**VERSION\_CODES**.**HONEYCOMB**)** **{**

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

view**.**removeOnLayoutChangeListener**(**oldValue**);**

**}**

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

view**.**addOnLayoutChangeListener**(**newValue**);**

**}**

**}}**

We want to remove the old one before we add a new one, but in this case, we don

Just using this API, Data Binding does the thinking for you. You just need to think about how you animate the change. Of course, you can also do this with multiple properties as well. We

### Depdendency Injection[(36:20)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,99,true);)

Let

**public** **interface** **TestableAdapter** **{**

**@BindingAdapter(**"android:src"**)**

**void** **setImageUrl(**ImageView imageView**,** String url**);}public** **interface** **DataBindingComponent** **{**

TestableAdapter **getTestableAdapter();}**DataBindingUtil**.**setDefaultComponent**(**myComponent**);**

or binding **=** MyLayoutBinding**.**inflate**(**layoutInflater**,** myComponent**);**

Obviously whatsetImageUrl. What if I have some state that I want to have in my BindingAdapter? Or, let

What we can do is create a binding component,DataBindingComponent, which is an interface. When you have an instance method, we

You can also do this on a per-layout basis. In this case, one sets the default and it can be used in all of your layouts. Then we know exactly what component to use to get your adapter.

You may also want to use your component as a parameter. For example, we just saw thissetImageURLbefore.

**@BindingAdapter(**"android:src"**)public** **static** **void** **setImageUrl(**MyAppComponent component**,**

ImageView view**,**

String imageUrl**)** **{**

component**.**getImageCache**().**loadInto**(**view**,** imageUrl**);}**

We want to use some kind of state. Let

### Event Handlers[(38:56)](https://academy.realm.io/posts/data-binding-android-boyar-mount/javascript:presentz.changeChapter(0,101,true);)

We have thisonClickattribute, we have aclickedmethod on handler.clickedcould begetClicked, orisClicked, or it could be a field,

<Button android:onClick="@{isAdmin ? handler.adminClick : handler.userClick}" />

*// No "setOnClick" method for View. Need a way to* **@BindingMethods({**

**@BindingMethod(**type **=** View**.**class**,**

attribute **=** "android:onClick"**,**

method **=** "setOnClickListener"**})** *// Look for setOnClickListener in View* **void** **setOnClickListener(**View**.**OnClickListener l**)**

*// Look for single abstract method in OnClickListener* **void** **onClick(**View v**);**

First of all, we need to find out whatonClickmeans? We knowonClickis notsetOnClick, because we looked and we saw that there was nosetOnClick, but thereonClickmeanssetOnClickListener. So we look atsetOnClickListener, which takes a parameter: it takes anonClickListenerargument, so let

We see that thereonClickListener, so we know that this could possibly be a listener that you want to use for your event handler. Now we look at the handler, and we find a method in there, calledclicked.

**static** **class** **OnClickListenerImpl1** **implements** OnClickListener **{**

**public** Handler mHandler**;**

**@Override**

**public** **void** **onClick(**android**.**view**.**View arg0**)** **{**

mHandler**.**adminClick**(**arg0**);**

**}}static** **class** **OnClickListenerImpl2** **implements** OnClickListener **{**

**public** Handler mHandler**;**

**@Override**

**public** **void** **onClick(**android**.**view**.**View arg0**)** **{**

mHandler**.**userClick**(**arg0**);**

**}}**

We found theclickedmethod, and it takes the same parameters. We have a match, yay! We know this is an event handler, so we know exactly what to do: we

So what do you do in the case of TextWatcher? Cause there is no single abstract method. There