本文的主题是ConstraintLayout。其实ConstraintLayout是Android Studio 2.2中主要的新增功能之一，也是Google在去年的I/O大会上重点宣传的一个功能。我们都知道，在传统的Android开发当中，界面基本都是靠编写XML代码完成的，虽然Android Studio也支持可视化的方式来编写界面，但是操作起来并不方便，我也一直都不推荐使用可视化的方式来编写Android应用程序的界面。

而ConstraintLayout就是为了解决这一现状而出现的。它和传统编写界面的方式恰恰相反，ConstraintLayout非常适合使用可视化的方式来编写界面，但并不太适合使用XML的方式来进行编写。当然，可视化操作的背后仍然还是使用的XML代码来实现的，只不过这些代码是由Android Studio根据我们的操作自动生成的。

另外，ConstraintLayout还有一个优点，它可以有效地解决布局嵌套过多的问题。我们平时编写界面，复杂的布局总会伴随着多层的嵌套，而嵌套越多，程序的性能也就越差。ConstraintLayout则是使用约束的方式来指定各个控件的位置和关系的，它有点类似于RelativeLayout，但远比RelativeLayout要更强大。

其实ConstraintLayout属于Android Studio 2.2的新特性，我在去年写《第二行代码》的时候就非常想要将这部分内容加入到新书里面，但是在尝试之后还是放弃了。因为ConstraintLayout的用法很多都是对控件进行拖拽，只用文字或者是一些静态图片实在太难将它的用法表达清楚了，因此不太适合写到书上。我当时的想法就是在博客上面写一篇ConstraintLayout的用法讲解，来弥补一下《第二行代码》中缺失的这部分新特性，那么今天这篇文章来了。

开始

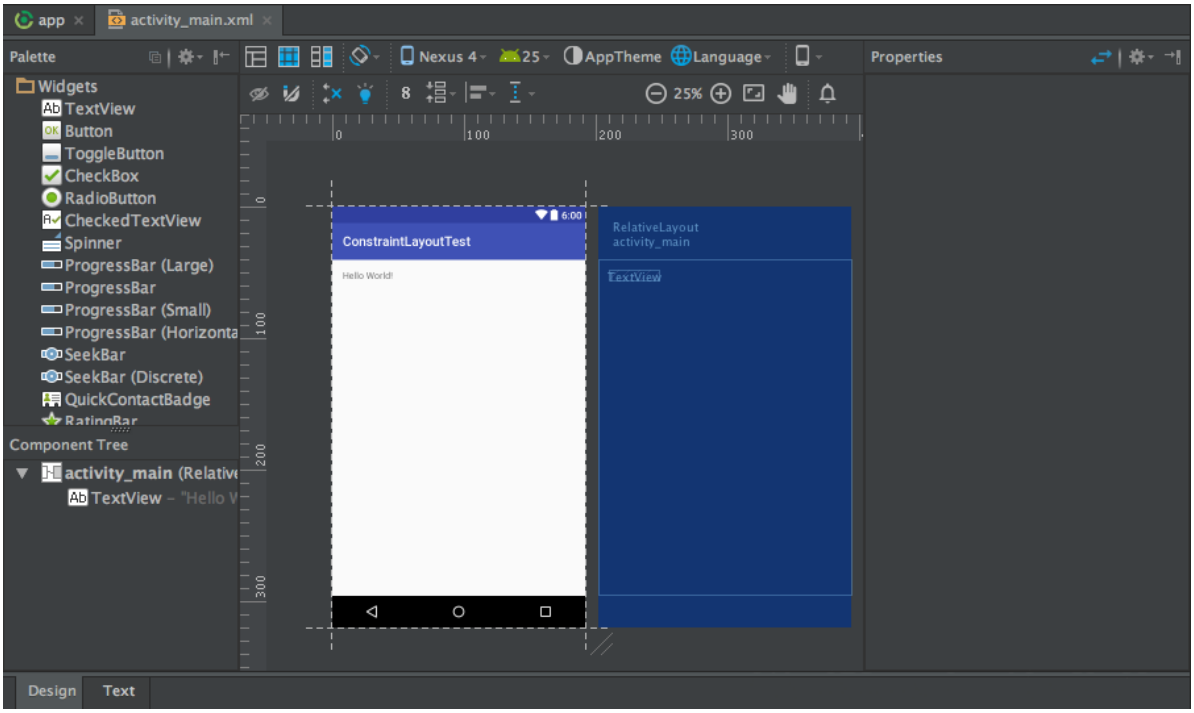
下面我们开始用边学边练的方式来进行学习，首先打开你的Android Studio，并新建一个ConstraintLayoutTest项目。另外，确保你的Android Studio是2.2或以上版本。

为了要使用ConstraintLayout，我们需要在app/build.gradle文件中添加ConstraintLayout的依赖，如下所示。

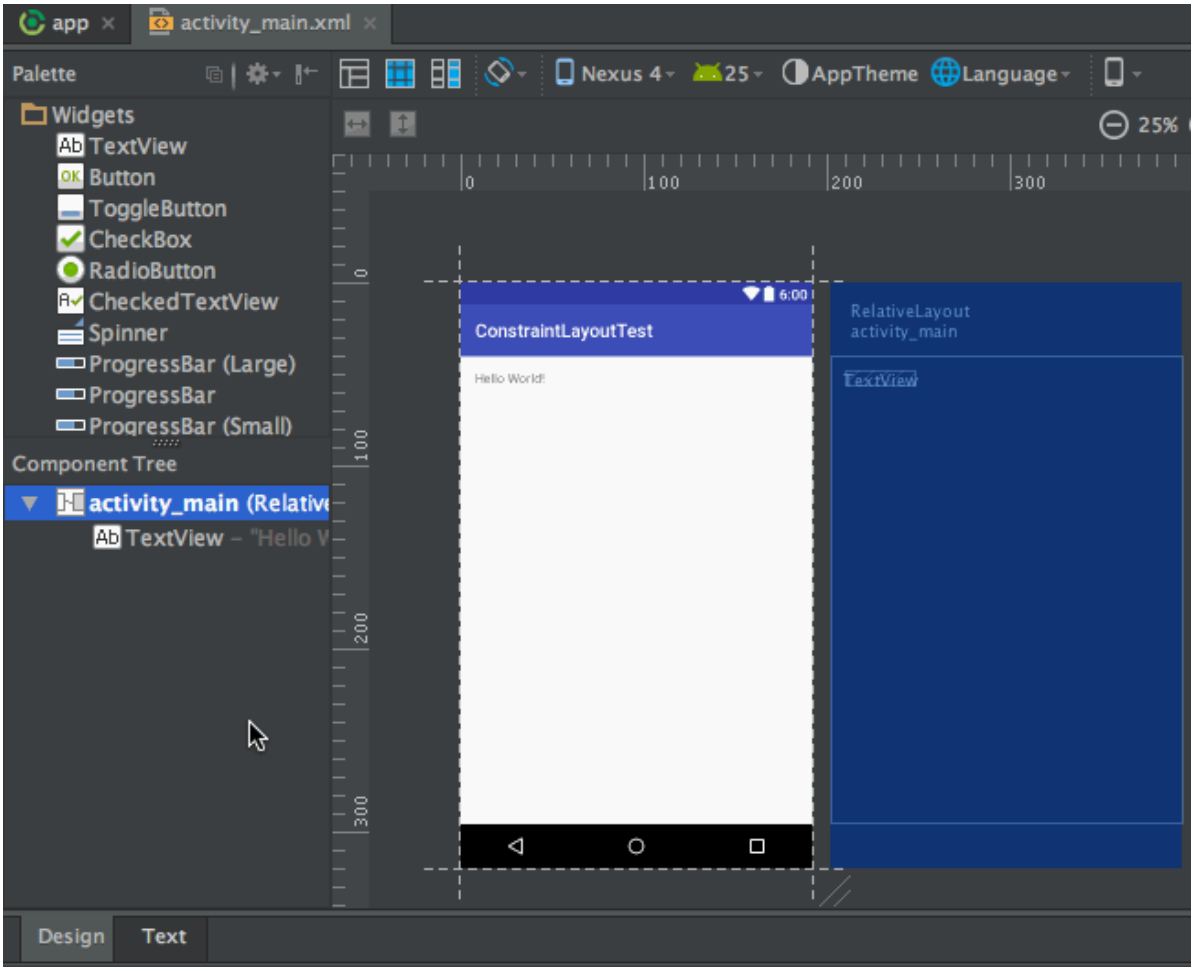
```
dependencies {
    compile 'com.android.support.constraint:constraint-layout:1.0.0-beta4'
}123
```

目前ConstraintLayout库最新的版本是1.0.0-beta4，还没有推出正式稳定版本，不过这并不影响我们提前进行学习和使用。

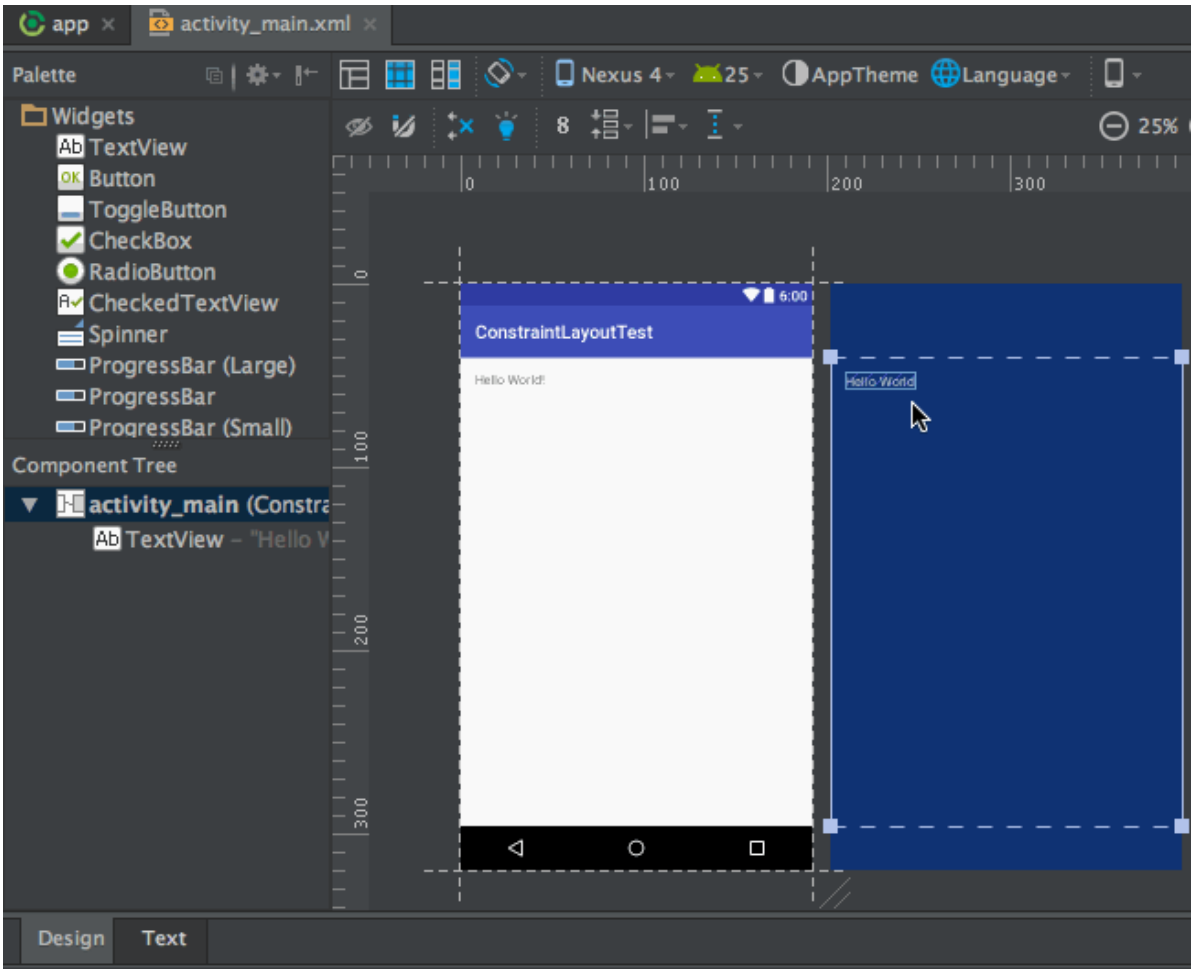
现在打开res/layout/activity_main.xml文件，由于这是一个新建的空项目，Android Studio会自动帮我们创建好一个布局，如下图所示。



不过，Android Studio自动创建的这个布局默认使用的是RelativeLayout，我们可以通过如下操作将它转换成ConstraintLayout。



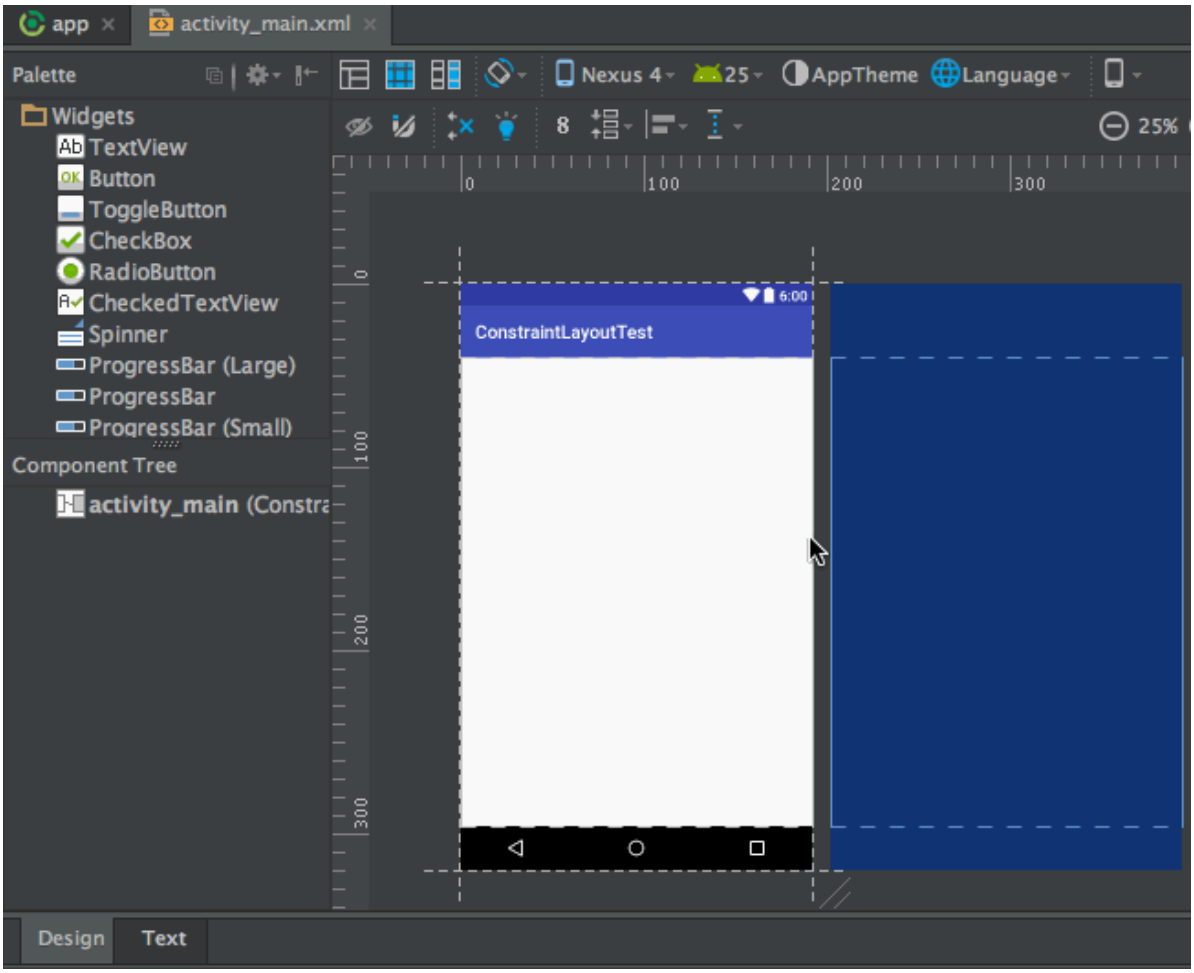
转换完成之后，原RelativeLayout中的内容也会自动转换到ConstraintLayout中，比如图中的TextView。如果你不需要它的话，可以选中这个控件，然后按键盘上的Delete键即可删除。



我们可以看到，现在主操作区域内有两个类似于手机屏幕的界面，左边的是预览界面，右边的是蓝图界面。这两部分都可以用于进行布局编辑工作，区别是左边部分主要用于预览最终的界面效果，右边部分主要用于观察界面内各个控件的约束情况。

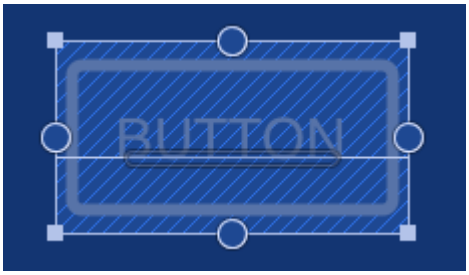
基本操作

下面我们来学习一些具体的操作吧，ConstraintLayout的基本用法很简单，比如我们想要向布局中添加一个按钮，那么只需要从左侧的Palette区域拖一个Button进去就可以了，如下图所示。

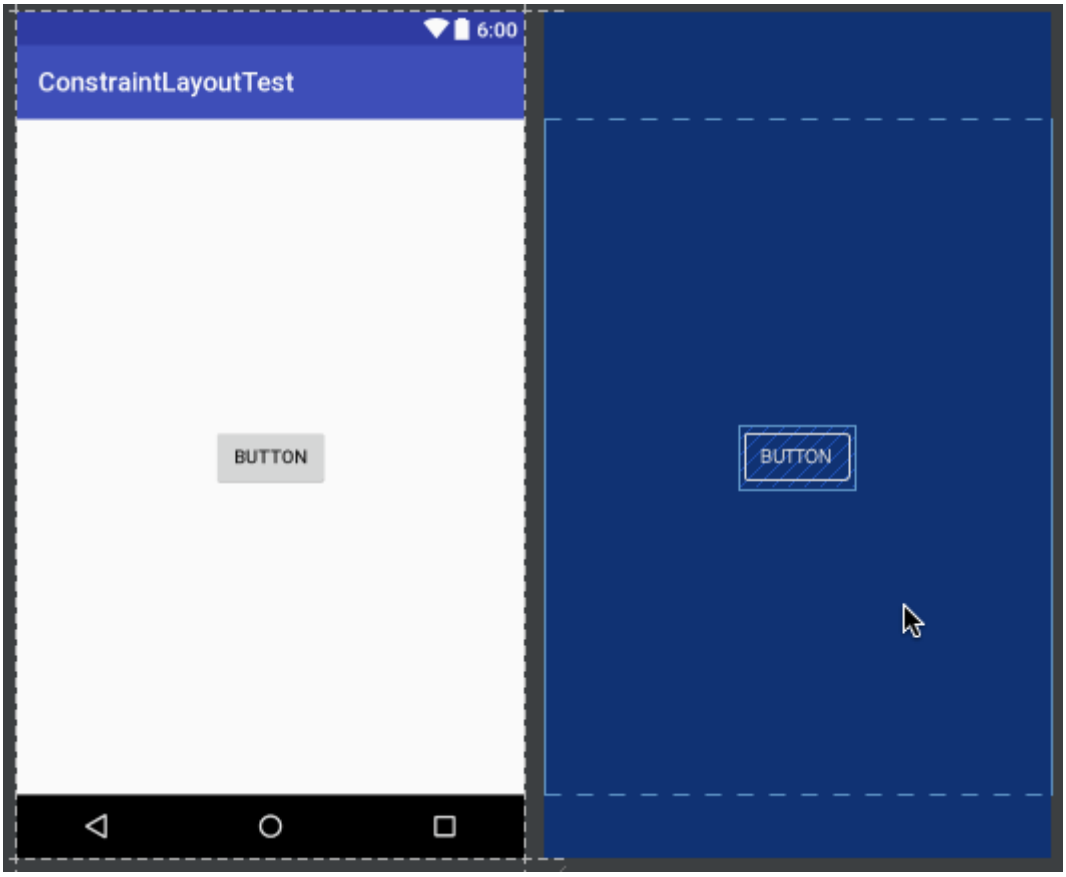


虽说现在Button已经添加到界面上了，但是由于我们还没有给Button添加任何的约束，因此Button并不知道自己应该出现在什么位置。现在我们在预览界面上看到的Button位置并不是它最终运行后的实际位置，如果一个控件没有添加任何约束的话，它在运行之后会自动位于界面的左上角。

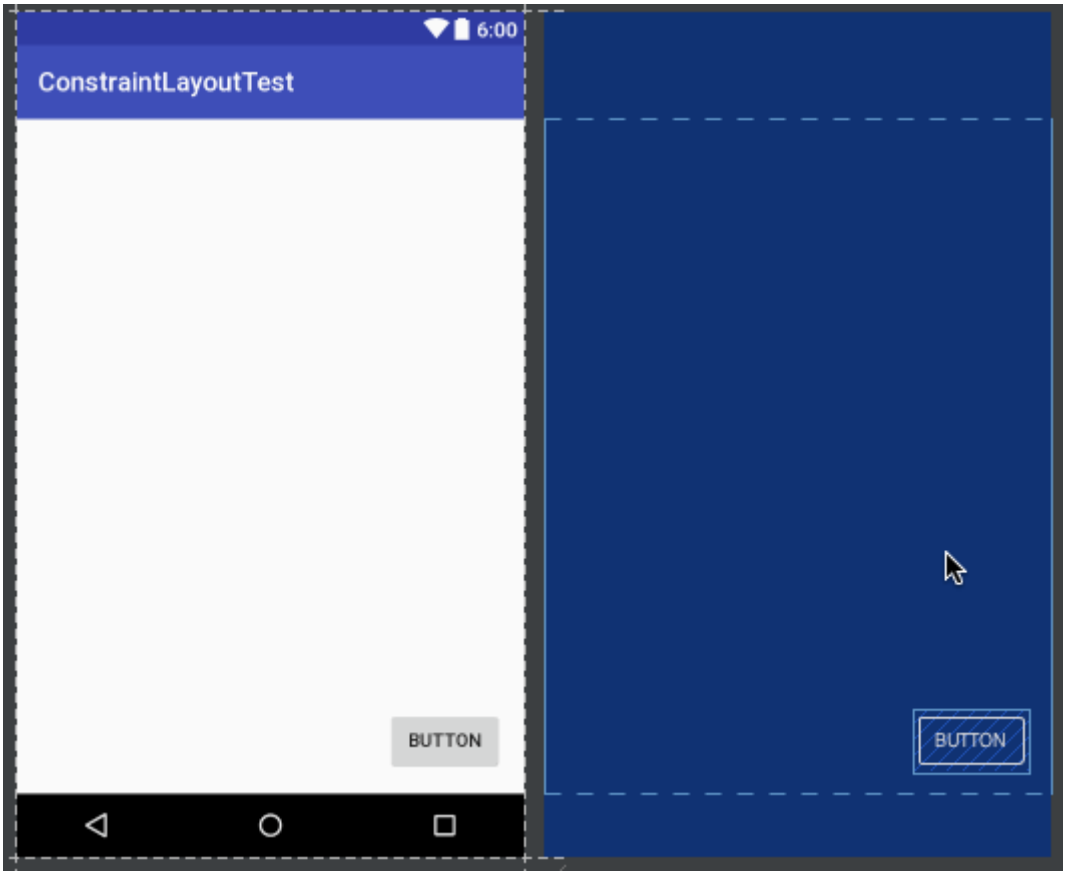
那么下面我们就来给Button添加约束，每个控件的约束都分为垂直和水平两类，一共可以在四个方向上给控件添加约束，如下图所示。



上图中Button的上下左右各有一个圆圈，这圆圈就是用来添加约束的，我们可以将约束添加到ConstraintLayout，也可以将约束添加到另一个控件。比如说，想让Button位于布局的右下角，就可以这样添加约束，如下图所示。

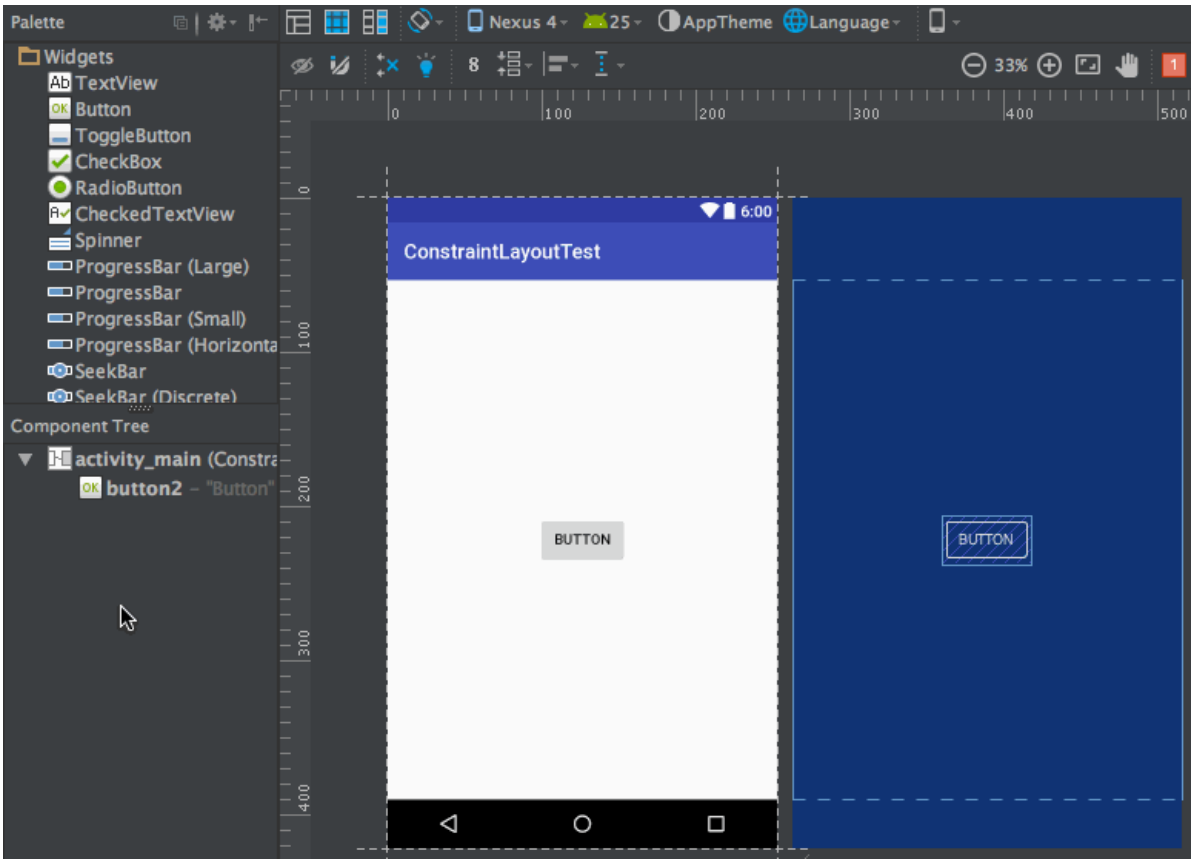


我们给Button的右边和下边添加了约束，因此Button就会将自己定位到布局的右下角了。类似地，如果我们想要让Button居中显示，那么就需要给它的上下左右都添加约束，如下图所示。

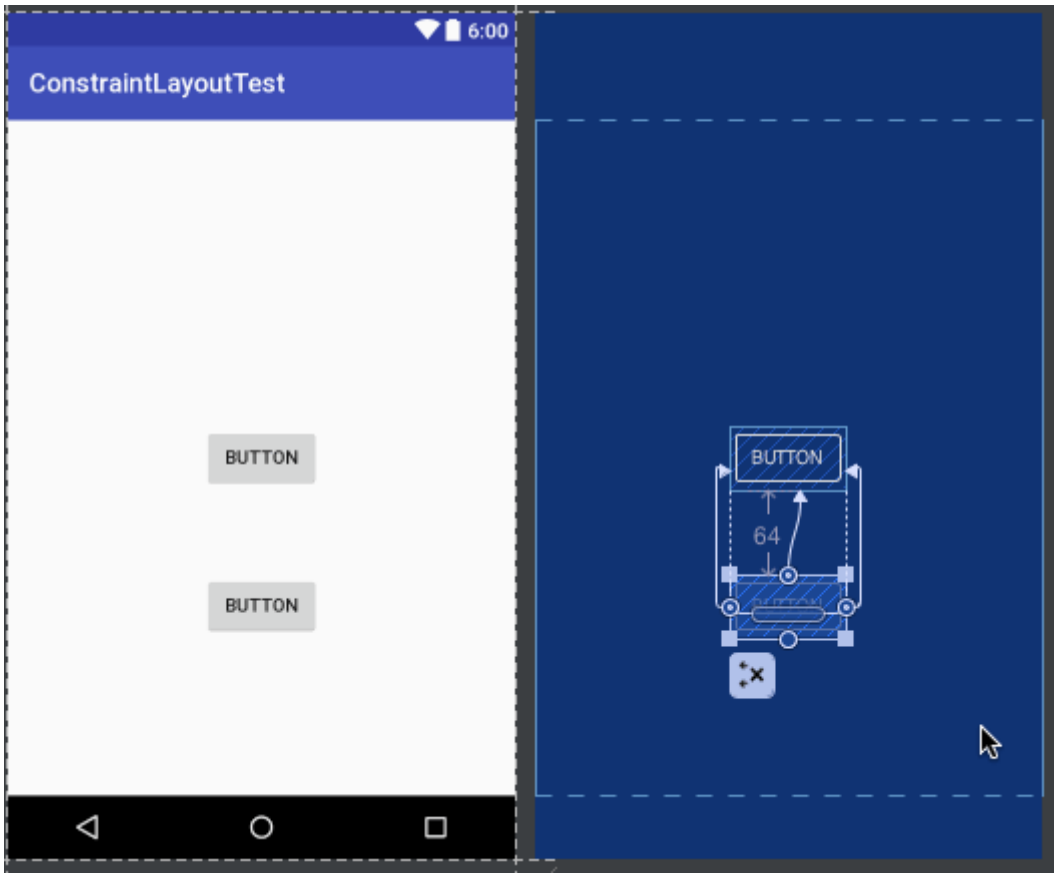


这就是添加约束最基本的用法了。

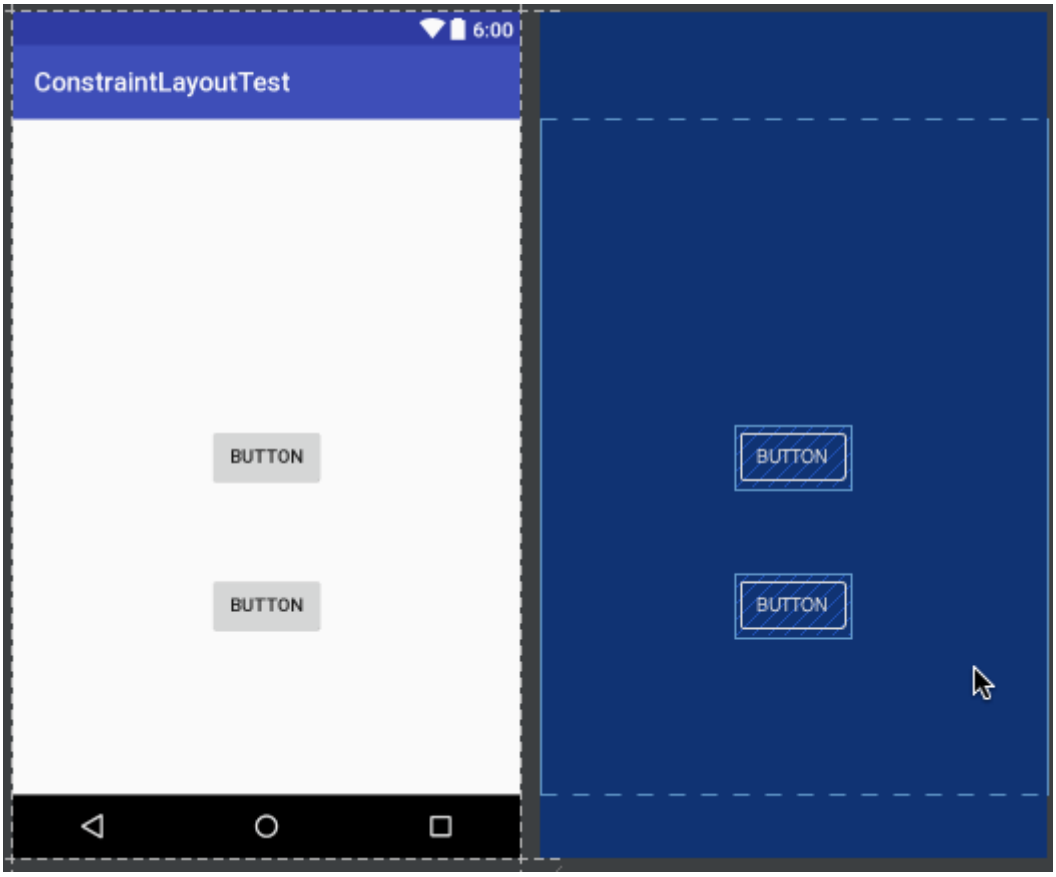
除此之外，我们还可以使用约束让一个控件相对于另一个控件进行定位。比如说，我们希望再添加一个Button，让它位于第一个Button的正下方，并且间距64dp，那么操作如下所示。



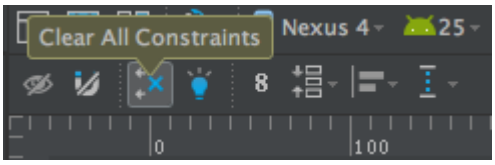
现在添加约束的方式我们已经学完了，那么该怎样删除约束呢？其实也很简单，删除约束的方式一共有三种，第一种用于删除一个单独的约束，将鼠标悬浮在某个约束的圆圈上，然后该圆圈会变成红色，这个时候单击一下就能删除了，如下图所示。



第二种用于删除某一个控件的所有约束，选中一个控件，然后它的左下角会出现一个删除约束的图标，点击该图标就能删除当前控件的所有约束了，如下所示。



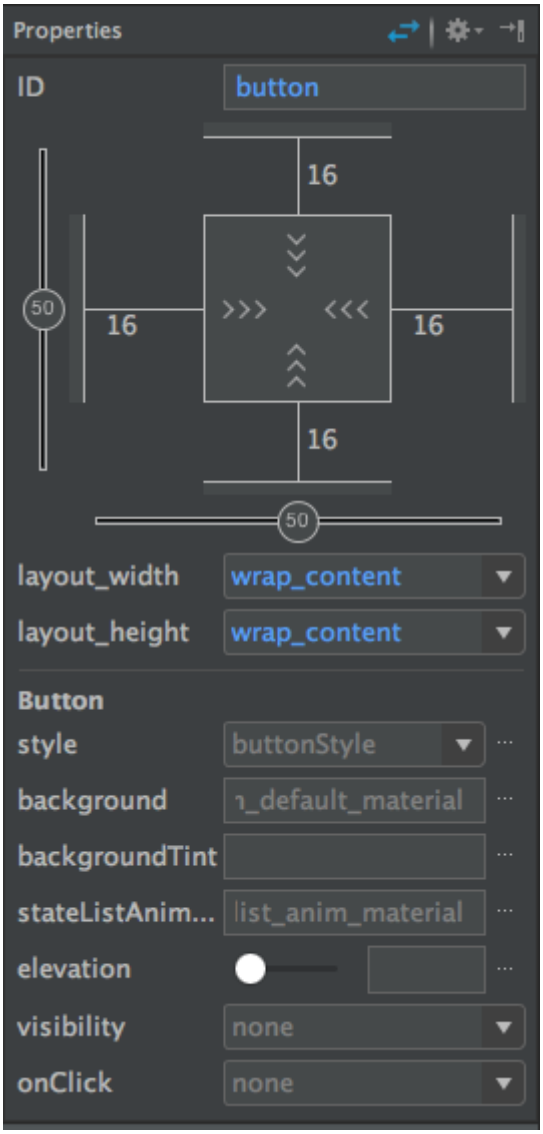
第三种用于删除当前界面中的所有约束，点击工具栏中的删除约束图标即可，如下图所示。



Inspector

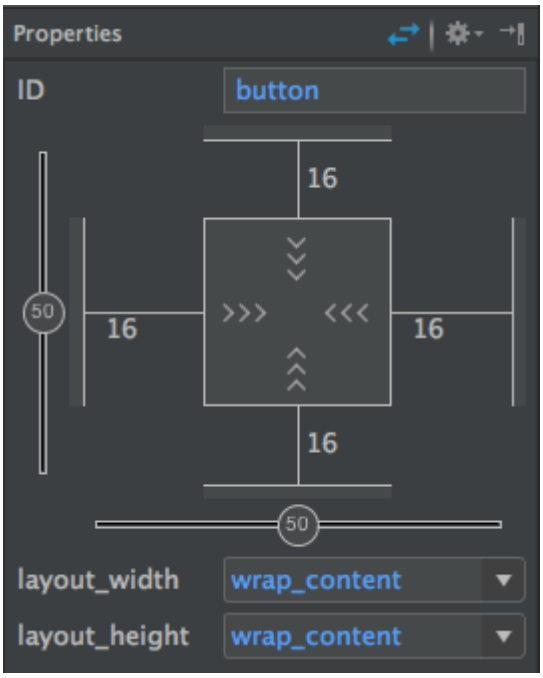
这样我们就把ConstraintLayout的基本用法学完了，接下来我们开始学习一些进阶的内容。

当你选中任意一个控件的时候，在右侧的Properties区域就会出现很多的属性选项，如下图所示。

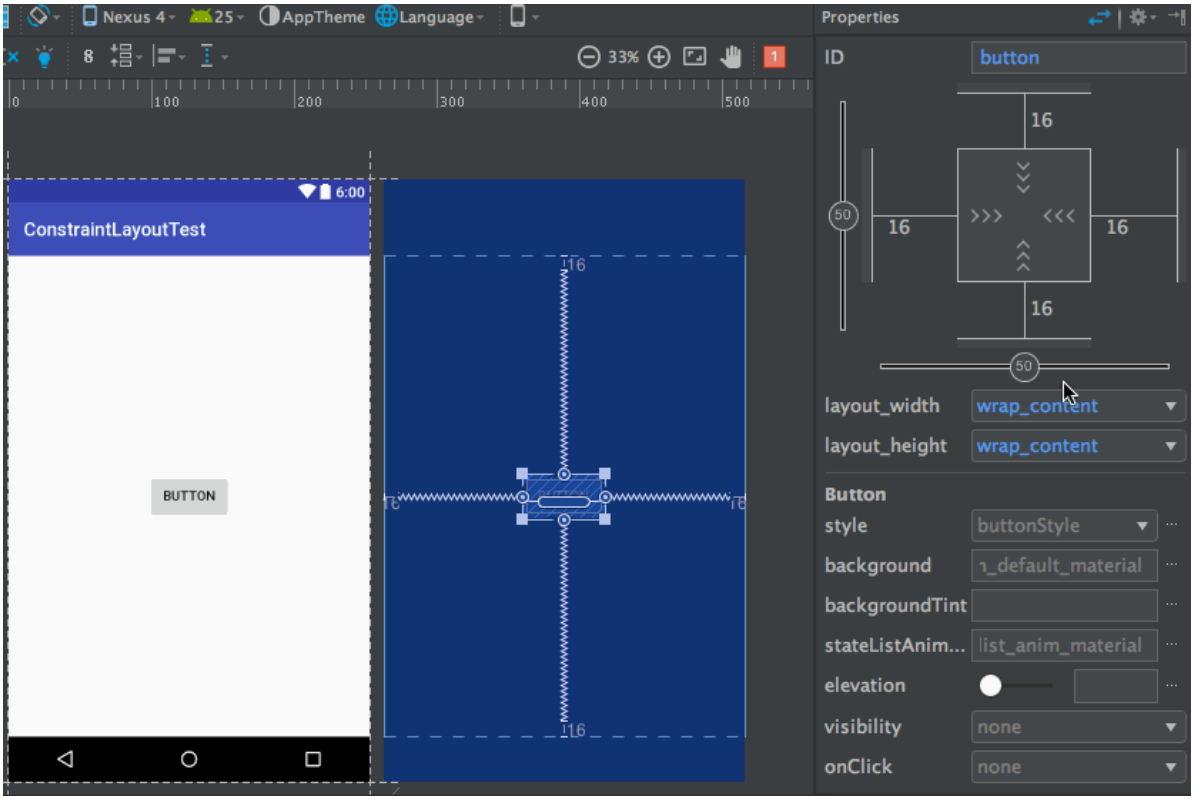


在这里我们就可以设置当前控件的所有属性，如文本内容、颜色、点击事件等等。这些功能都非常简单，我就不再进行详细介绍，大家自己点一点就会操作了。

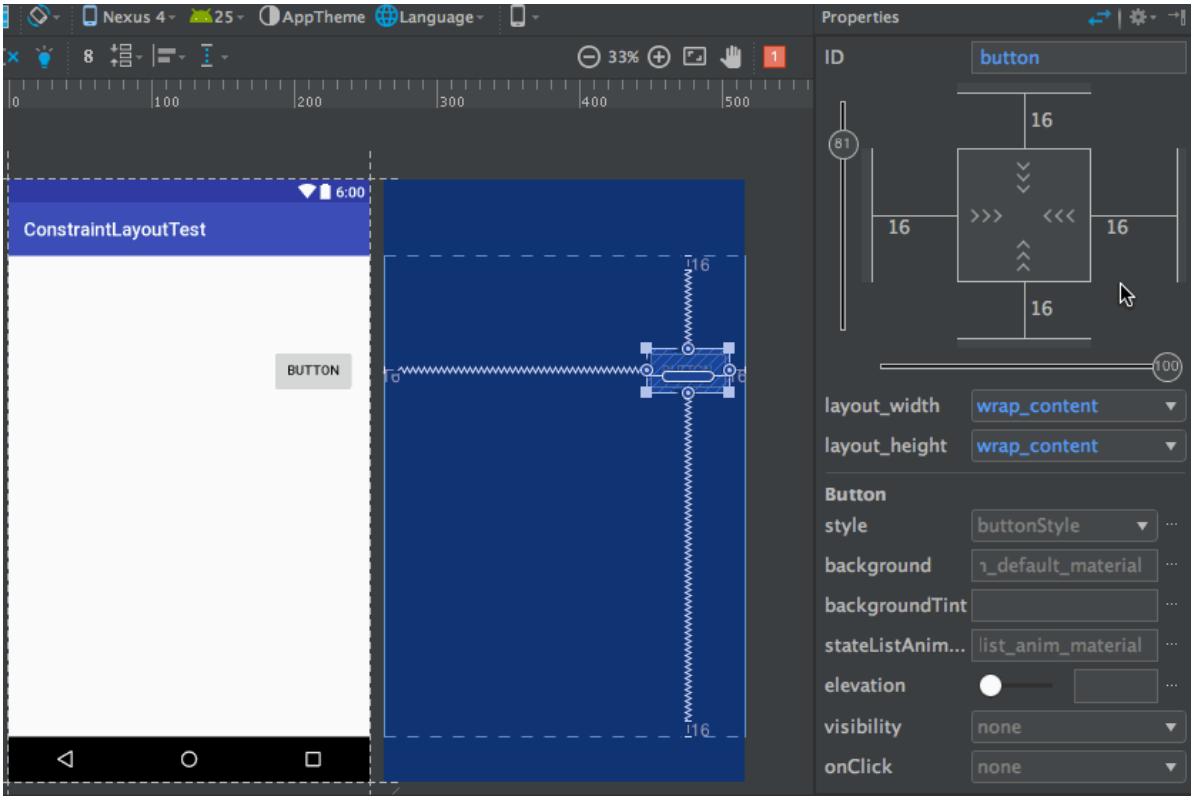
需要我们重点掌握的是Properties区域的上半部分，这部分也被称为Inspector。



首先可以看到，在Inspector中有一个纵向的轴和一个横向的轴，这两个轴也是用于确定控件的位置的。我们刚才给Button的上下左右各添加了一个约束，然后Button就能居中显示了，其实就是因为这里纵横轴的值都是50。如果调整了纵横轴的比例，那么Button的位置也会随之改变，如下图所示。



不过，虽然我们将横轴的值拖动到了100，但是Button并没有紧贴到布局的最右侧，这是为什么呢？实际上，Android Studio给控件的每个方向上的约束都默认添加了一个16dp的间距，从Inspector上面也可以明显地看出来这些间距的值。如果这些默认值并不是你想要的，可以直接在Inspector上进行修改，如下图所示：



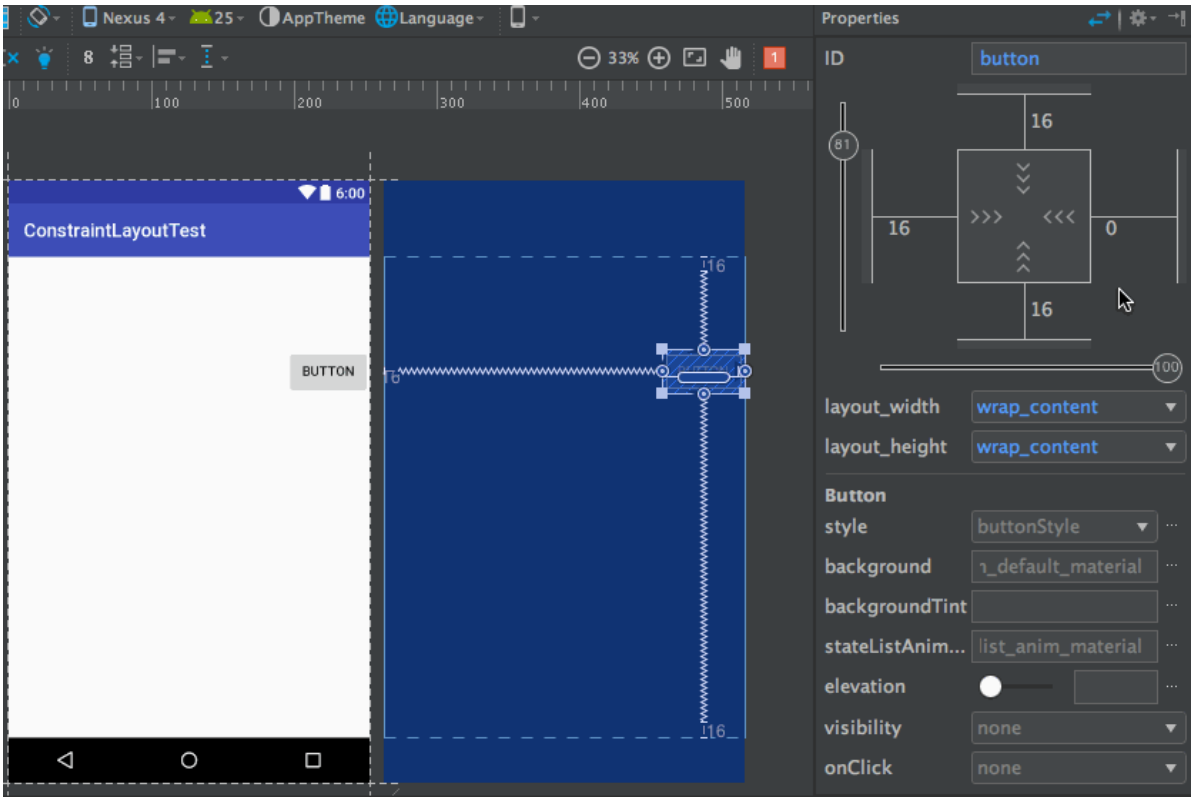
可以看到，修改成0之后Button右侧的间距就没了。

接下来我们再来学习一下位于Inspector最中间的那个正方形区域，它是用来控制控件大小的。一共有三种模式可选，每种模式都使用了一种不同的符号表示，点击符号即可进行切换。

- >>> 表示wrap content，这个我们很熟悉了，不需要进行什么解释。
- |—| 表示固定值，也就是给控件指定了一个固定的长度或者宽度值。
- /\/\ 表示any size，它有点类似于match parent，但和match parent并不一样，是属于ConstraintLayout中特有的一种大小控制方式，下面我们来重点讲解一下。

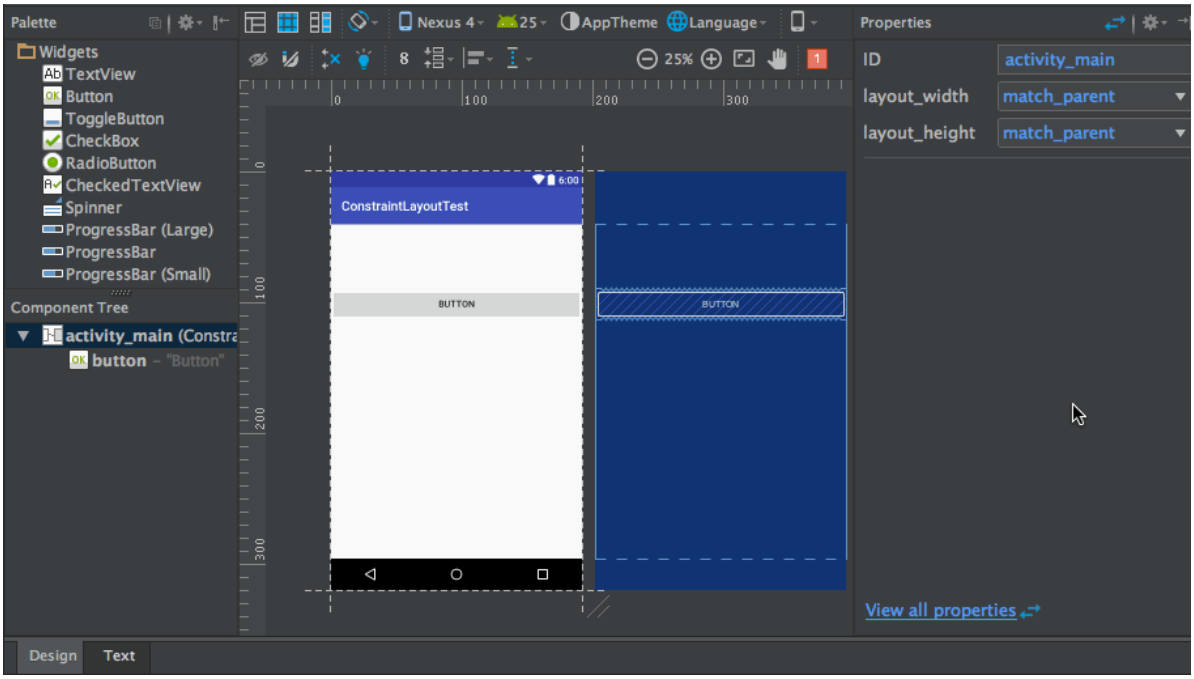
首先需要说明，在ConstraintLayout中是有match parent的，只不过用的比较少，因为ConstraintLayout的一大特点就是为了解决布局嵌套，既然没有了布局嵌套，那么match parent也就没有多大意义了。

而any size就是用于在ConstraintLayout中顶替match parent的，先看一下我们怎样使用any size实现和match parent同样的效果吧。比如说我想让Button的宽度充满整个布局，操作如下图所示。



可以看到，我们将Button的宽度指定成any size，它就会自动充满整个布局了。当然还要记得将Button左侧的间距设置成0才行。

那有的朋友可能会问了，这和match parent有什么区别呢？其实最大的区别在于，match parent是用于填充满当前控件的父布局，而any size是用于填充满当前控件的约束规则。举个例子更好理解，如果我们有一个新的Button，它的其中一个约束是添加到当前这个Button上的，那么any size的效果也会发生改变，如下图所示。



通过上图的演示，相信你已经很好地理解any size的作用了。

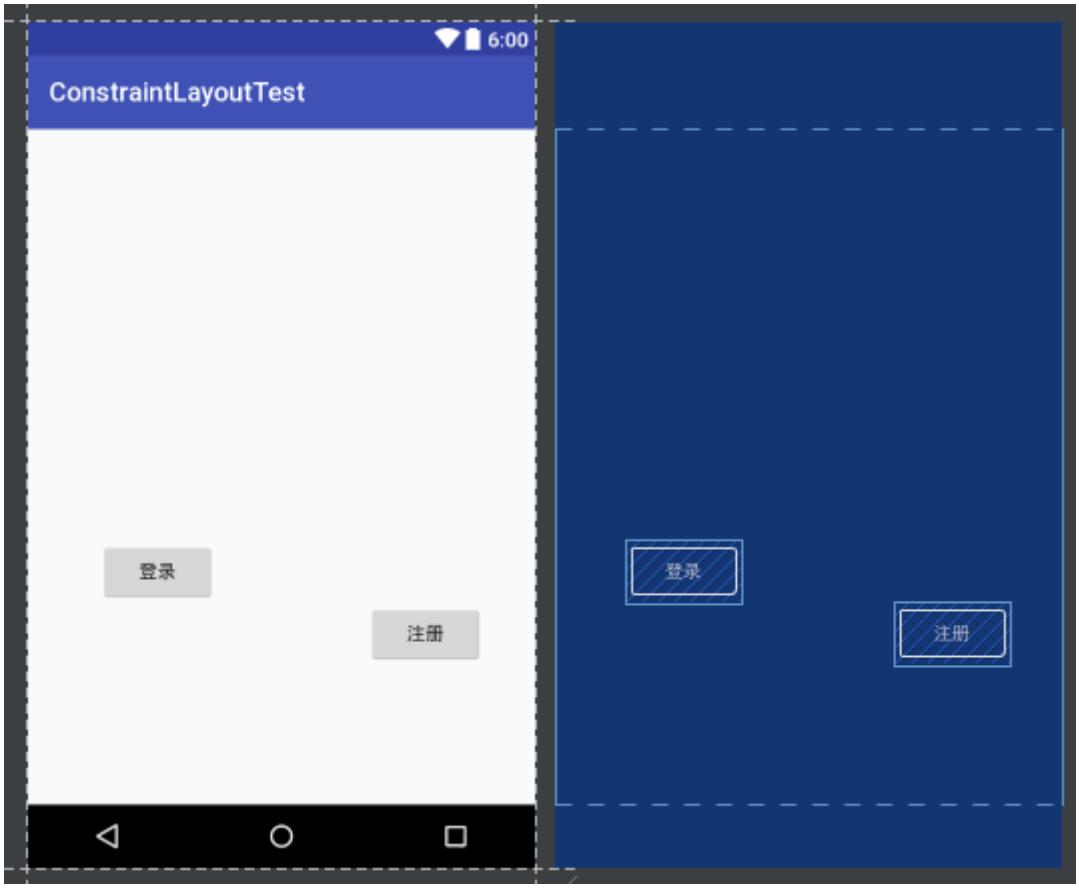
Guidelines

现在你已经对ConstraintLayout比较熟悉，并且能使用ConstraintLayout来编写一些简单的界面了。不过目前有一个问题可能还比较头疼，刚才我们已经实现了让一个按钮居中对齐的功能，如果我们想让两个按钮共同居中对齐该怎么实现呢？

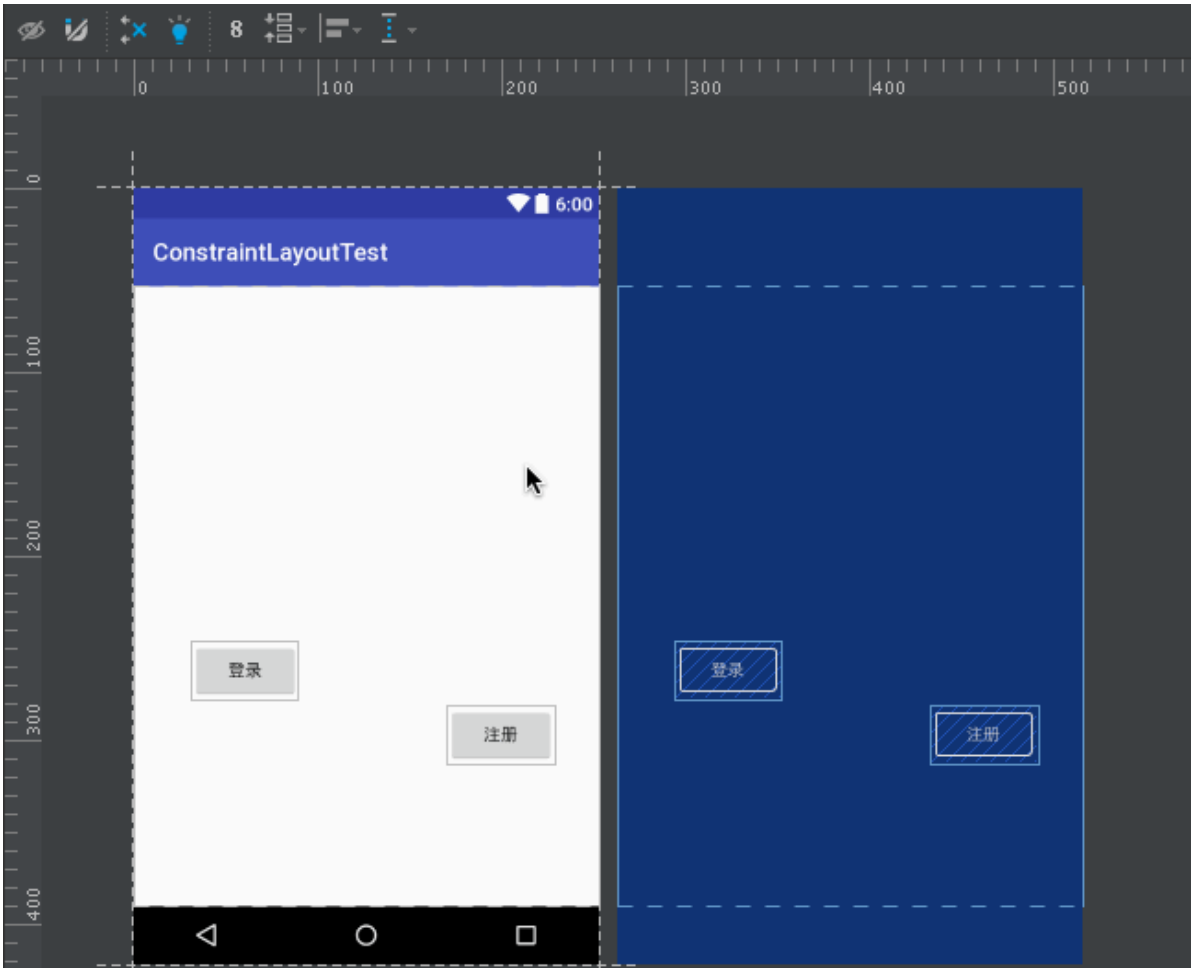
其实这个需求很常见，比如说在应用的登录界面，都会有一个登录按钮和一个注册按钮，不管它们是水平居中也好还是垂直居中也好，但肯定都是两个按钮共同居中的。

想要实现这个功能，仅仅用我们刚刚学的那些知识是不够的，这需要用到ConstraintLayout中的一个新的功能，Guidelines。

下面我们还是通过实际操作来学习一下Guidelines的用法吧。比如现在已经向界面中添加了登录和注册这两个按钮，如下图所示。

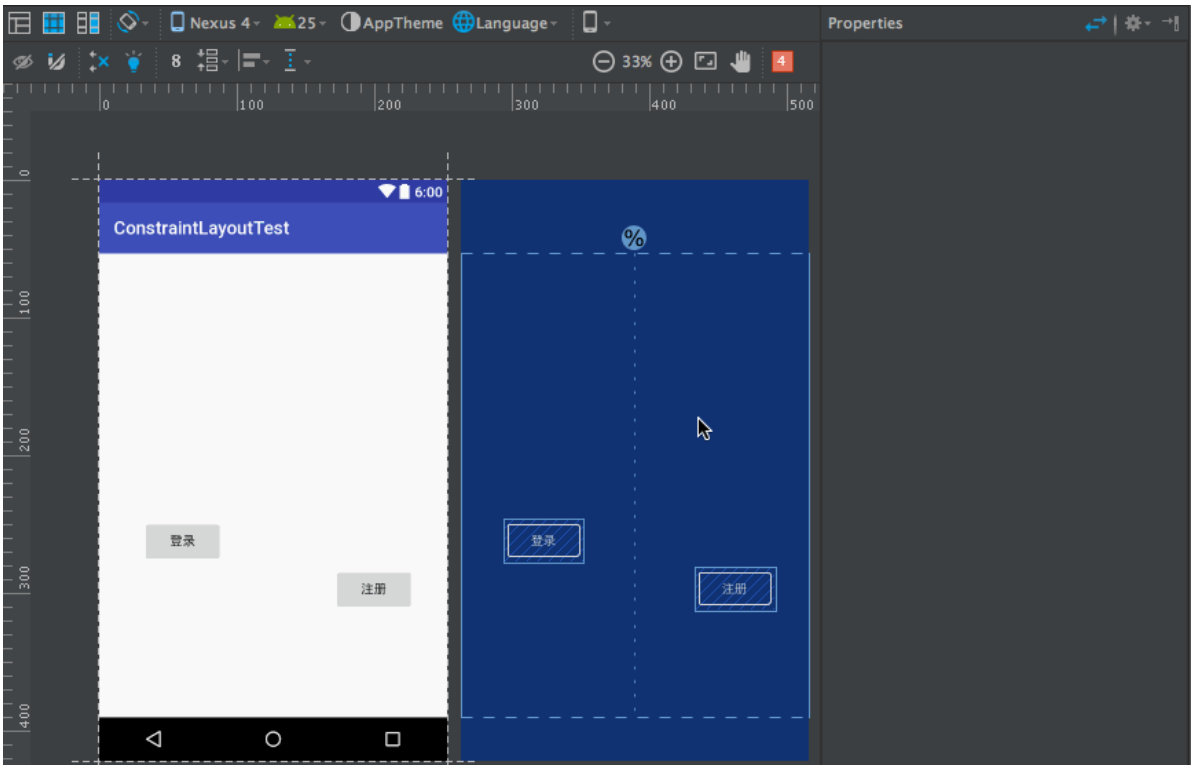


然后我们希望让这两个按钮在水平方向上居中显示，在垂直方向上都距离底部64dp，那么就需要先添加一个垂直方向上的Guideline，如下图所示。



我来对上图中的操作进行一下解释。首先点击通知栏中的Guidelines图标可以添加一个垂直或水平方向上的Guideline，这里我们需要的是垂直方向上的。而Guideline默认是使用的dp尺，我们需要选中Guideline，并点击一下最上面的箭头图标将它改成百分比尺，然后将垂直方向上的Guideline调整到50%的位置，这样就将准备工作做好了。

接下来我们开始实现让两个按钮在水平方向上居中显示，并距离底部64dp的功能，如下图所示。



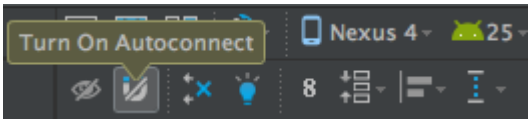
可以看到，我们给登录按钮的右边向Guideline添加约束，登录按钮的下面向底部添加约束，并拖动按钮让它距离底部64dp。然后给注册按钮的左边向Guideline添加约束，注册按钮的下面向登录按钮的下面添加约束。这样就实现了让两个按钮在水平方向上居中显示，在垂直方向上都距离底部64dp的功能了。

自动添加约束

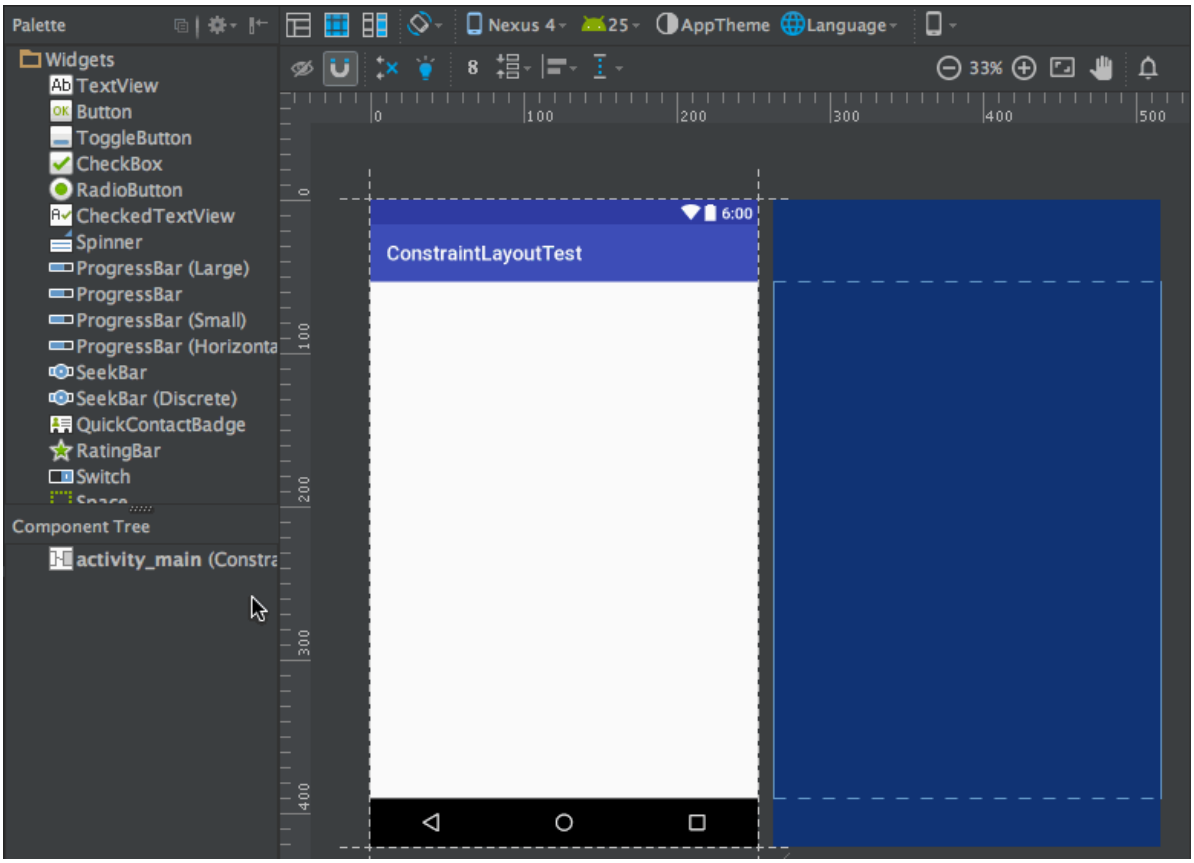
不过如果界面中的内容变得复杂起来，给每个控件一个个地添加约束也是一件很繁琐的事情。为此，ConstraintLayout中支持自动添加约束的功能，可以极大程度上简化那些繁琐的操作。

自动添加约束的方式主要有两种，一种叫Autoconnect，一种叫Inference，我们先来看第一种。

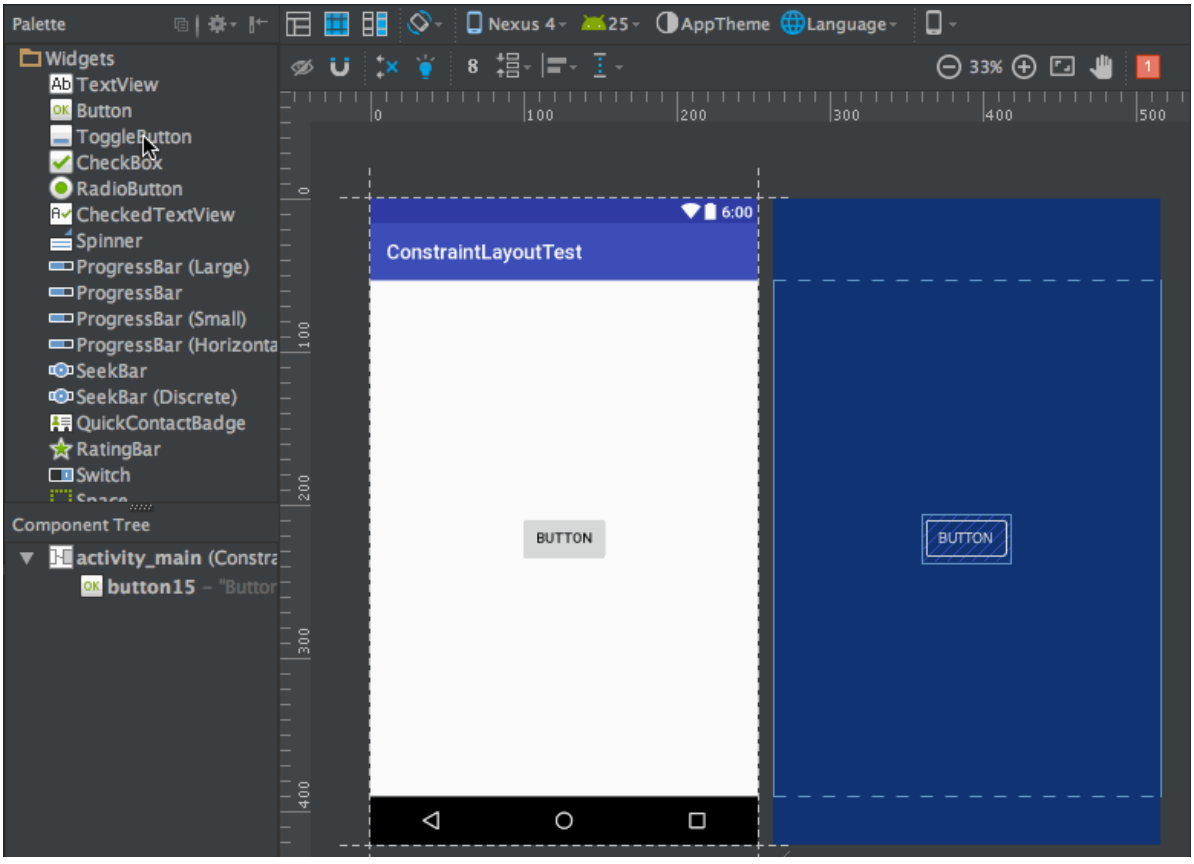
想要使用Autoconnect，首先需要在工具栏中将这个功能启用，默认情况下Autoconnect是不启用的，如下图所示。



Autoconnect可以根据我们拖放控件的状态自动判断应该如何添加约束，比如我们将Button放到界面的正中央，那么它的上下左右都会自动地添加上约束，如下图所示。



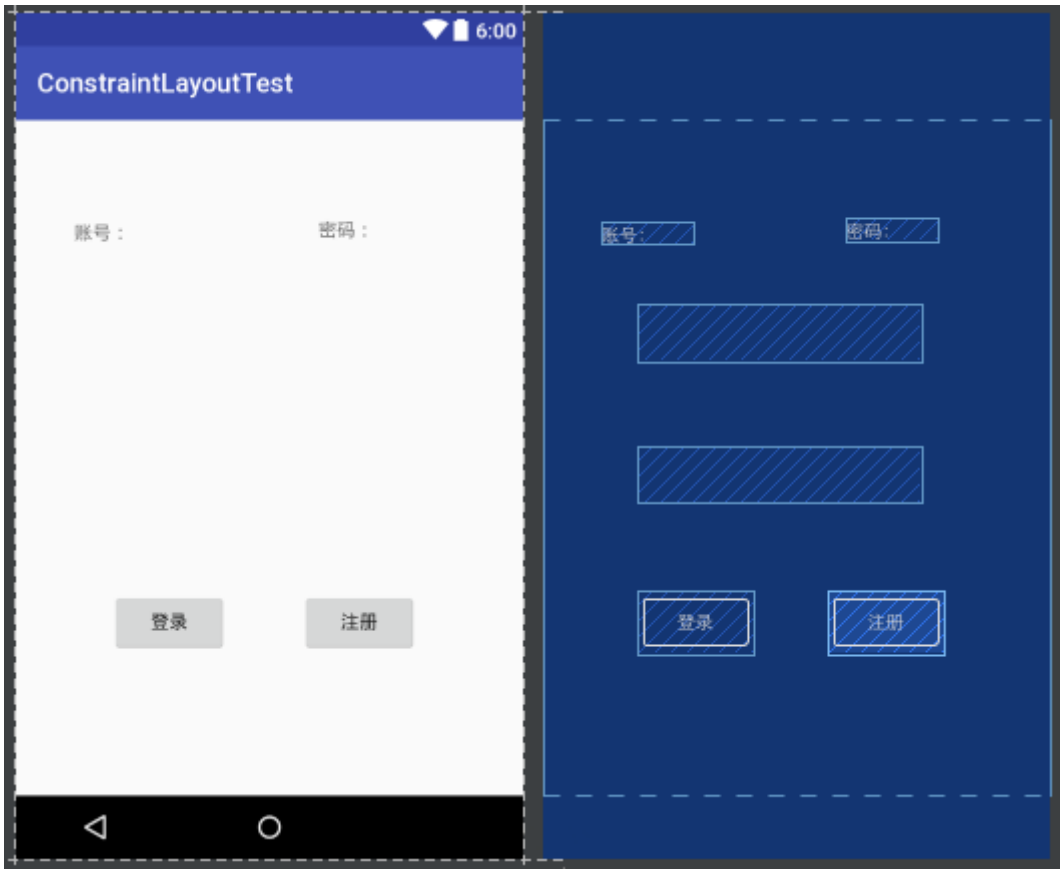
然后我们在这个Button的下方再放置一个Button，效果如下。



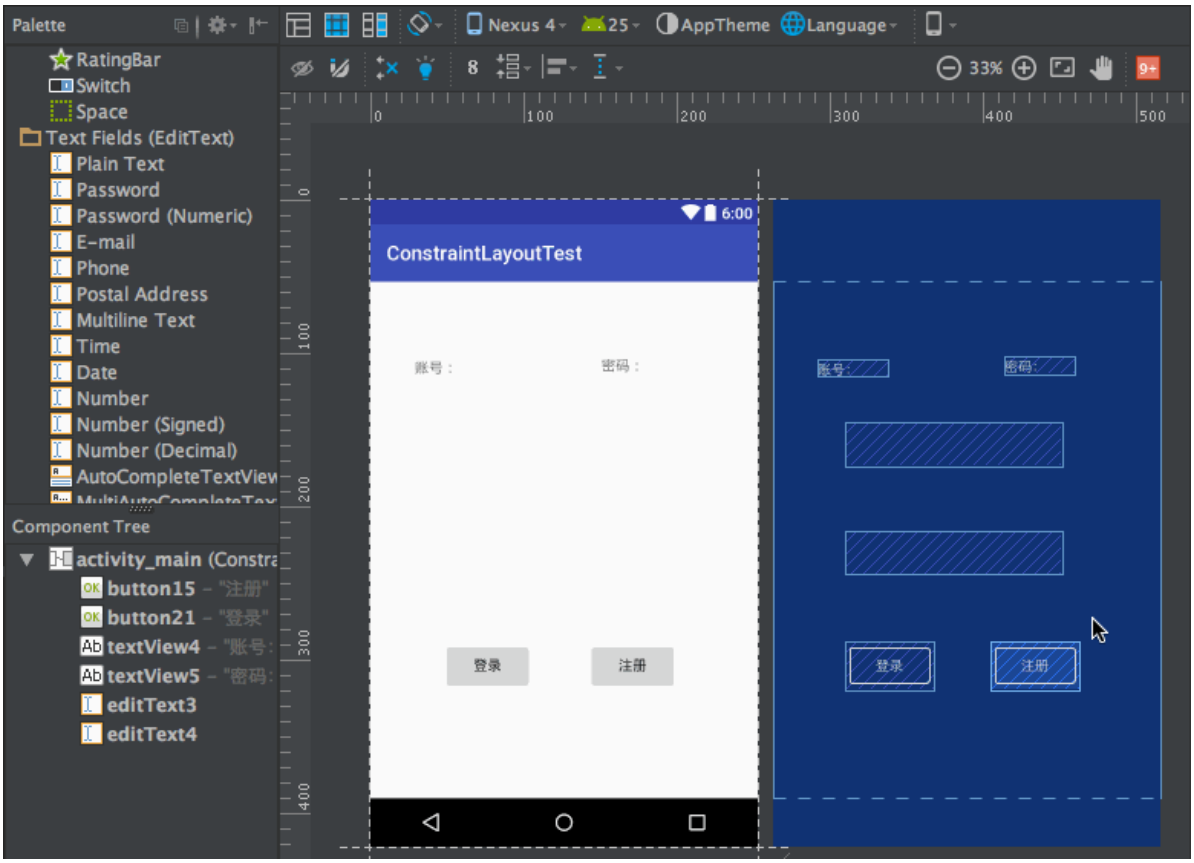
可以看到，只需要将Button拖放到界面上，Autoconnect会判断我们的意图，并自动给控件添加约束。不过Autoconnect是无法保证百分百准确判断出我们的意图的，如果自动添加的约束并不是你想要的话，还可以在任何时候进行手动修改。总之，可以把它当成一个辅助工具，但不能完全靠它去添加控件的约束。

以上是Autoconnect的用法，接下来我们看一下Inference的用法。Inference也是用于自动添加约束的，但它比Autoconnect的功能要更为强大，因为AutoConnect只能给当前操作的控件自动添加约束，而Inference会给当前界面中的所有元素自动添加约束。因而Inference比较适合用来实现复杂度比较高的界面，它可以一键自动生成所有的约束。

下面我们就通过一个例子来演示一下Inference的用法，比如界面上现在有两个TextView，两个EditText，和两个Button，如下图所示。



接下来我们先将各个控件按照界面设计的位置进行摆放，摆放完成之后点击一下工具栏上的Infer Constraints按钮，就能为所有控件自动添加约束了，如下图所示。



现在运行一下程序，最终效果如下图所示：