

2018

# Android 移动开发实战

程源

广东机电职业技术学院

2018/9/1

## 目录

2.1 本章目标 .....	2 -
2.2 创建 NEC Vocab 项目 .....	3 -
2.3 用户登录界面设计 .....	6 -
2.3.1 布局的概念 .....	6 -
2.3.2 图形布局工具 .....	8 -
2.3.3 ConstraintLayout 的引入 .....	8 -
2.3.4 组件 .....	10 -
2.3.5 字符串资源 .....	14 -
2.4 从布局 XML 到视图对象 .....	17 -
2.4.1 资源与资源 ID .....	18 -
2.4.2 组件的引用 .....	19 -
2.4.3 监听器的设置 .....	21 -
2.5 创建水平模式布局 .....	23 -
2.5.1 设备配置与备选资源 .....	23 -
2.5.2 创建水平模式布局 .....	24 -
2.6 跟踪 Activity 的生命周期 .....	26 -
2.6.1 利用日志跟踪 Activity 生命周期 .....	28 -
2.6.2 使用 LogCat .....	30 -

## 2 实现用户登录界面

### 2.1 本章目标

本讲开始我们将通过开发一款“新观念”英语词汇学习—NEC Vocab 应用，开启读者的 Android 移动开发旅程。NEC Vocab 是一个具有文字、图片、音频、视频和语音功能的个性化的英语词汇学习 APP。

通过学习开发 NEC Vocab 应用，读者不仅可以掌握 Android 开发的重要理论知识，而且可以获得实际的开发经验和开发技巧。

这当然不是简单的任务，不可能一蹴而就，我们将用接下来的所有篇幅来实现它。个性化的学习 APP 不能不存储用户信息，因此，本讲将从实现最基本的用户**登录**模块的开始（图 2-1），介绍编写 Android 应用需要掌握的一些新的概念。

学完本次课，如果没能理解全部内容，也不必担心。后续课程还会有更加详细的讲解，我们将再次温习并理解这些概念。



图 2-1 本讲将实现的 NEC Vocab 的登录界面

## 2.2 创建 NEC Vocab 项目

我们首先创建 NEC Vocab 项目。

启动 Android Studio。选择 **File→New Project...**。可以看到新建项目向导。在该向导的 **Applicable name** (应用名称) 处输入 NEC Vocab, 作为项目名称 (见图 2-2)。在 **Company Domain** (公司域名) 处输入 aime.studio.com(这里输入你自己的公司域名或根据你的喜好起名)。这样做后, 可以看到 **Package name** (包名) 自动变成 com.studio.aime.necvocab。对于 **Project Location**, 可以使用任意位置。

注意, 包名遵循了“DNS”反转约定, 亦即将企业、组织或公司的域名反转后, 在尾部附上项目名称。遵循此约定可以保证包名的唯一性。这样, 同一设备和 Google Play 商店的各类应用就可以区分开来了。

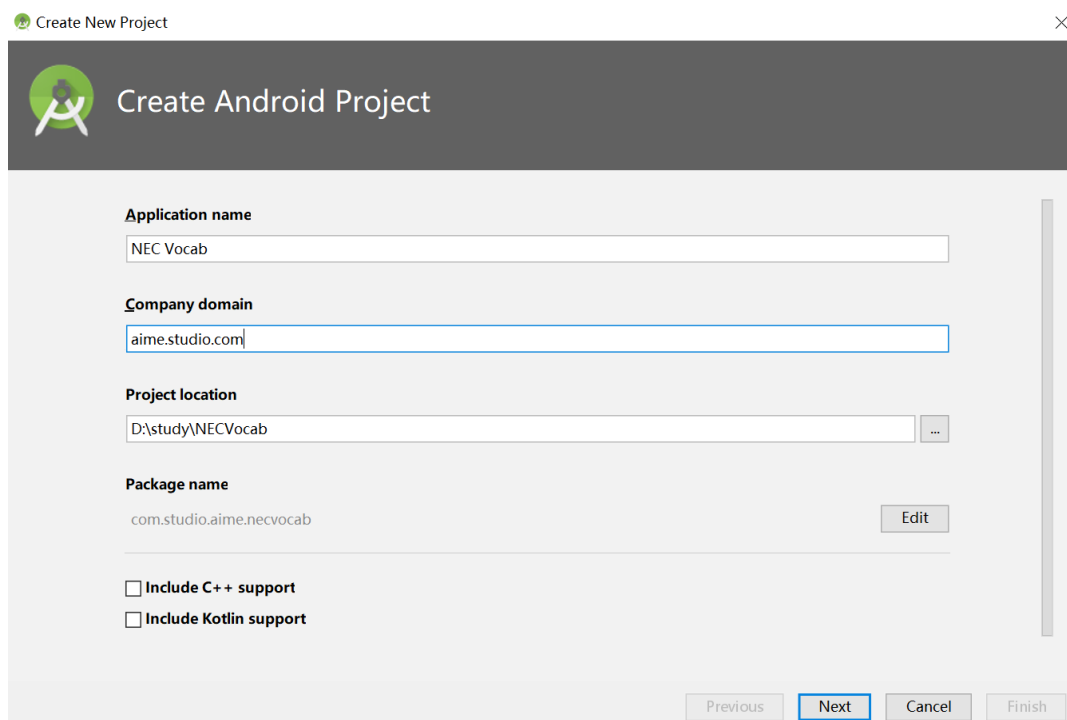


图 2-2 创建新 Android 项目

点击 **Next**, 下一个屏幕用来配置应用如何与不同版本的 Android 设备适配。NEC Vocab 应用支持智能手机和平板, 所以只需选择 **Phone and Tablet**。选择最低支持 SDK (Minimum SDK) 版本, API21:Android 5.0(Lollipop), 见图 2-3。Android 开发工具每年会更新多次, 因此当前的向导可能会与本讲义略有不同。

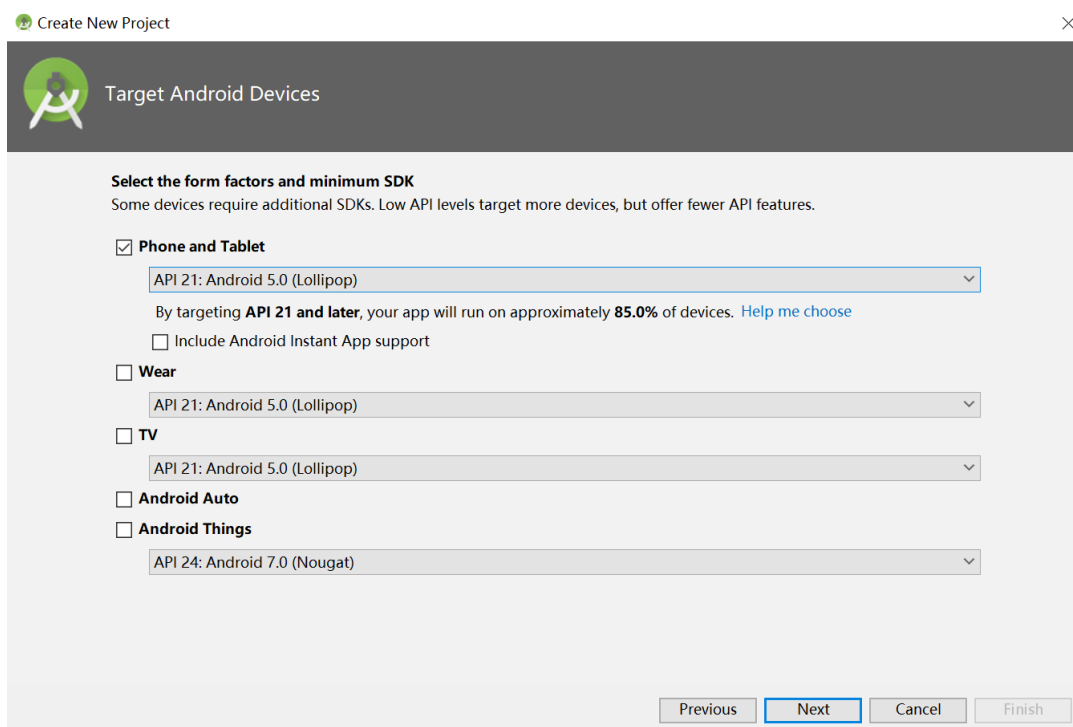


图 2-3 确定支持 NEC Vocab 应用的装置

点击 **Next**，系统会提示为应用 NEC Vocab 选择一个模板（图 2-4）。这里我们选择 **Empty Activity**。

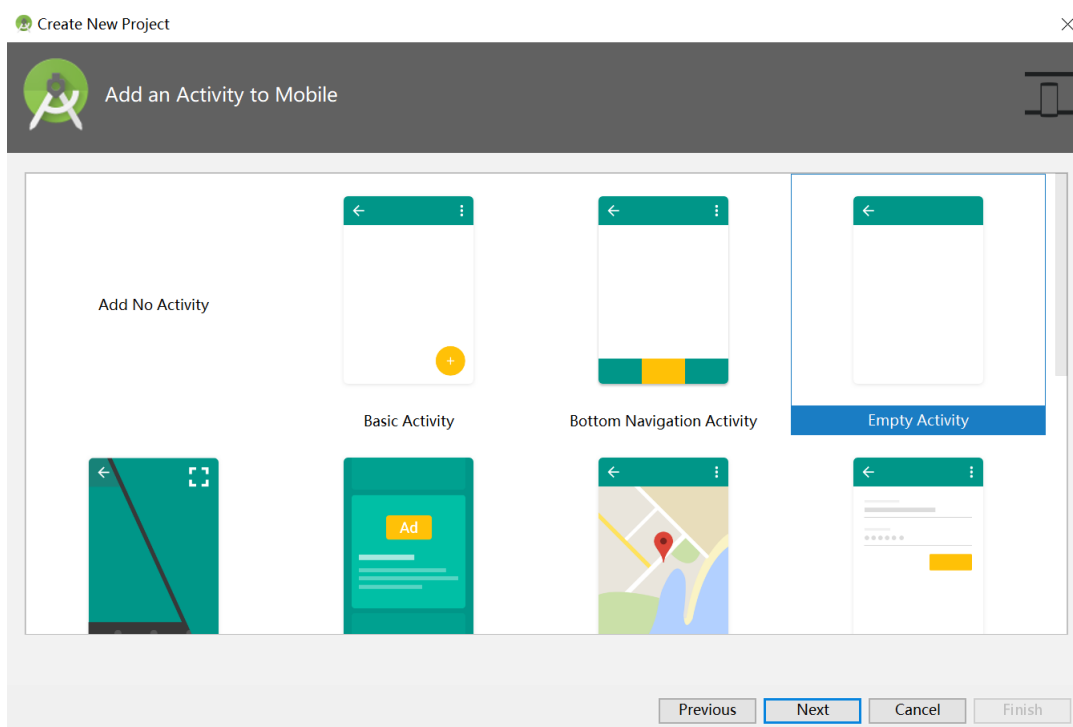


图 2-4 选择 Activity 类型

点击 **Next**，在应用向导的最后一个窗口，将 **Activity Name** 命名为：LoginActivity。如图 2-5 所示。注意子类名的 Activity 后缀。尽管不是必须的，但建议遵循这一命名约定。

这里保留了在 **Generate Layout File** 和 **Backwards Compatibility (AppCompat)** 两个复选框上的勾选。前者将同时创建一个布局文件，这里是 activity\_login.xml 文件；后者将使 LoginActivity 类继承自 AppCompatActivity 类。

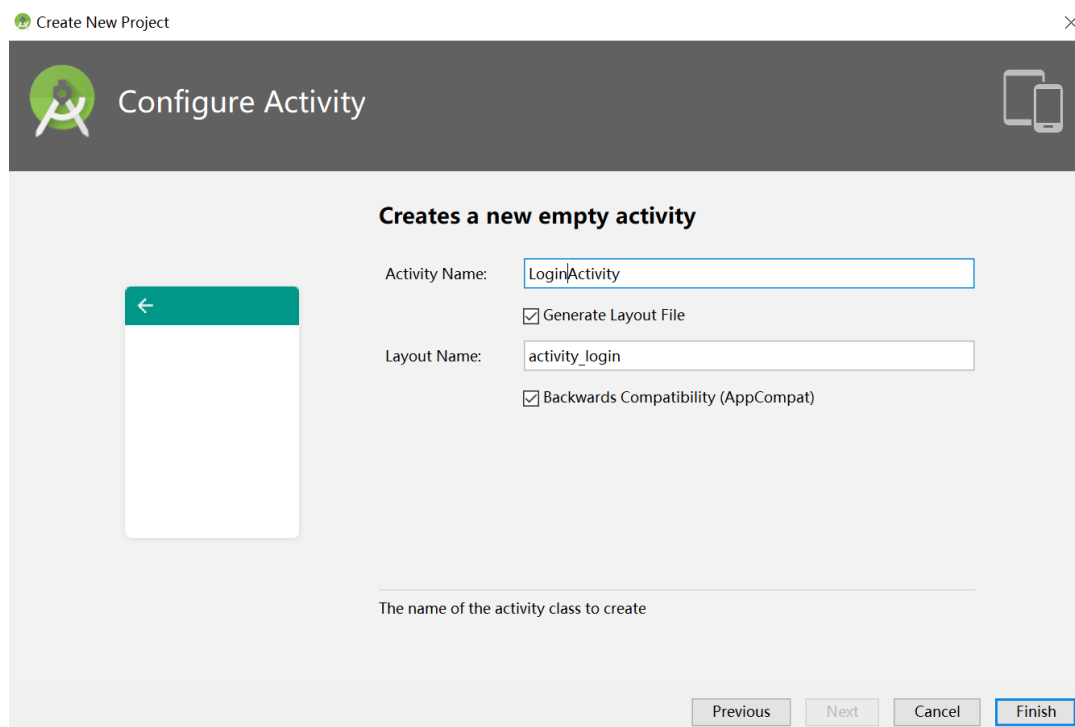
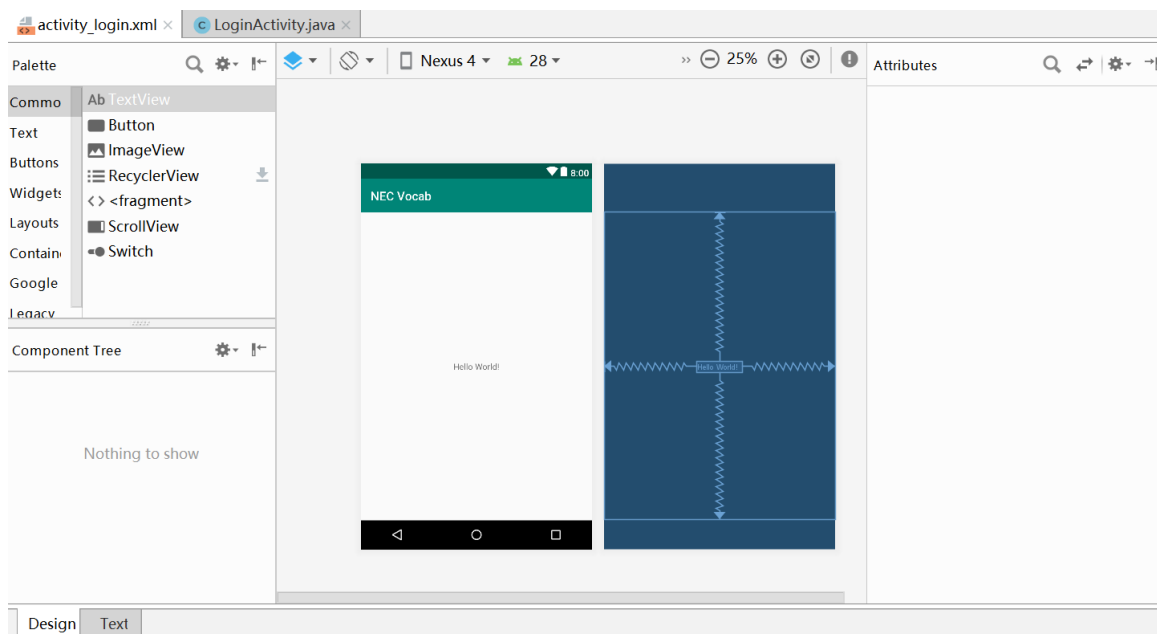



图 2-5 配置新的 Activity

可以看到 **Layout Name** 自动更新为 activity\_login，体现布局与 activity 的对应关系。布局的命名规则是：将 activity 名称的单词顺序倒过来并全部转换为小写字母，然后在单词间添加下划线。

点击 **Finish**，Android Studio 将创建并打开你的新的项目，如图 2-6 所示。这里 Android Studio 编辑器已经打开了 LoginActivity.java 文件。这是向导为我们创建的 java 文件。同时在编辑区右上角还可以看到另一个名为 activity\_login.xml 的选项卡。单击 activity\_login.xml 的选项卡便可以看到向导为我们创建的如图 2-6 所示的默认的布局文件。



2-6 activity\_login.xml 文件的默认布局

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（如果没有显示蓝图界面，可以如图 2-6 所示，从上边的  下拉菜单中选中“Design+Blueprint”）。这两部分都可以用于布局编辑工作，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。

如果想在编辑器中看到代码，可单击底部的“Text”标签页。

## 2.3 用户登录界面设计

Android 开发通常从 UI 界面（User Interface）入手。UI 界面是 Android 系统的重要组成部分，是人机交互的重要窗口，是用户对一款应用的第一印象。

### 2.3.1 布局的概念

Android 程序中的界面是通过布局（Layout）文件设定的。每个 Activity 创建时都会默认包含一个主界面布局文件，该文件位于 `res/layout` 目录中。

**Layout**（布局）定义了一系列用户界面对象以及它们显示在屏幕上的位置。组成 layout 的定义保存在后最为 xml 的文件中。每个定义用来创建屏幕上的一个对象，如按钮、文本或图片

信息。

NEC Vocab 应用的登录界面包含一个名为 `activity_login.xml` 的布局文件。目前 `activity_login.xml` 定义了默认的布局。Android 的默认设置更改频繁，但 xml 内容应当和代码清单 2-1 类似。

#### 代码清单 2-1 默认的布局文件（`activity_login.xml`）

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

</android.support.constraint.ConstraintLayout>
```

默认的布局默认定义了两个组件：**ConstraintLayout** 和 **TextView**。

组件是组成用户界面的构造模块。组件可以显示文字或图像、与用户交互，甚至是布置屏幕上的其他组件。按钮、文本输入控件和选择框等都是组件。

Android SDK 内置了多种组件，通过配置各种组件可以获得所需的用户界面及行为。每一个组件是 **View** 类或其子类（如 **TextView**、**EditText** 或 **Button**）的一个具体实例。

不过，图 2-6 所示的默认组件不是我们需要的，**LoginActivity** 需要如图 2-1 所示的用户界面。

**LoginActivity** 的界面将包含下列五个组件：

- 一个 **ConstraintLayout** 组件；
- 两个 **EditText** 组件；
- 两个 **Button** 组件。



## 2.3.2 图形布局工具

在 2016 年 Google I/O 推出的**约束布局**（ConstraintLayout）之前，Android 的布局主要以手动输入 XML 的方式创建。图形布局工具的功能较弱，只起一定的辅助作用。约束布局的推出使图形布局工具的功能变得非常强大，足以满足日常绝大多数布局的开发。本书的布局设计将主要使用图形布局工具。

打开 activity\_login.xml 布局文件，然后选择窗口底部的 **Design** 标签页。

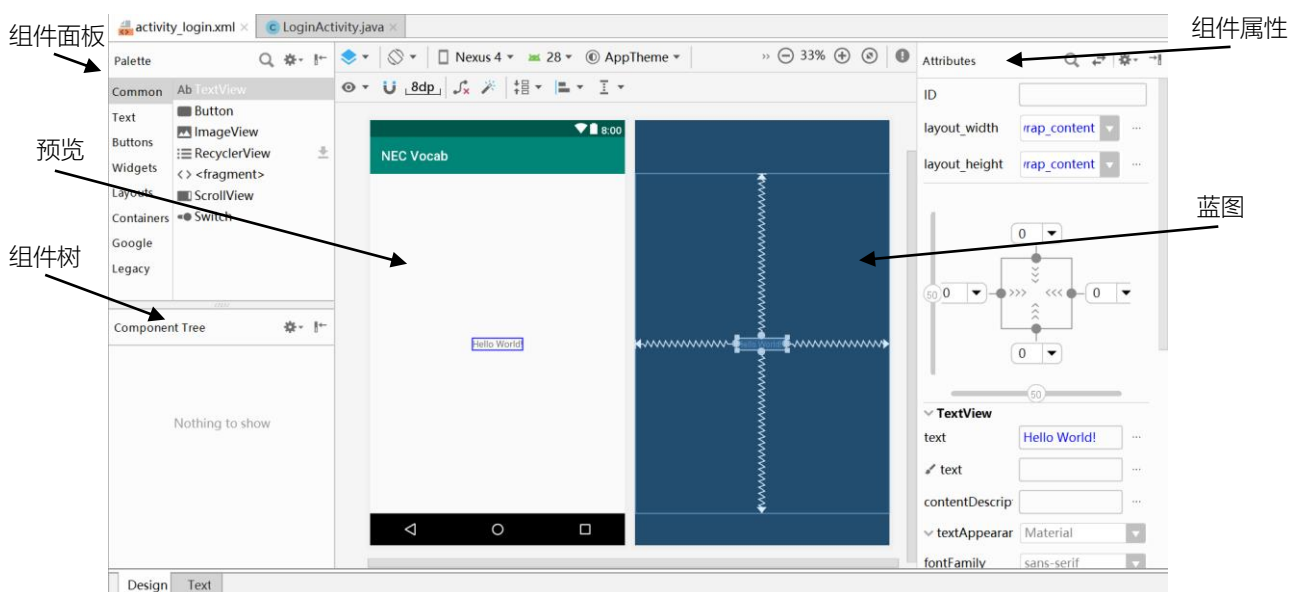


图 2-7 图形布局工具视图

如图 2-7 所示，图形布局工具的中间区域是布局界面的**预览**窗口。右边紧挨着的是**蓝图**（blueprint）视图。蓝图和预览视图有点像，但它能显示各个组件视图的轮廓。预览视图让你看到布局长什么样，而从蓝图可以看出各个组件视图的大小和比例。

图形布局工具界面左侧是**组件面板**视图，它包含了按类别组织的所有你可能用到的组件。

图形布局工具界面的左下方是**组件树**，组件树表明了布局是如何组织组件的。

图形布局工具界面的右边是**组件属性**视图。在此视图中，你可以查看并编辑组件树中已选中的组件的属性。

## 2.3.3 ConstraintLayout 的引入

从代码清单 2-1 可以看到 activity\_login.xml 文件的**根布局**（最外层组件）是 **ConstraintLayout**。

**约束布局** ConstraintLayout 是扁平式的布局方式，无任何嵌套。构建 ConstraintLayout 版本的布局可以仅仅使用布局编辑器，而不需要手工编辑 XML。从支持力度来看，将成为主流布局样式，完全代替其他布局。

## 使用 ConstraintLayout

使用 ConstraintLayout 需要给布局添加一系列**约束**。可以把约束想象为弹簧。它会向中间拉拢分别系在两头的东西。例如，如图 2-8 所示的 TextView 视图左右两边到其父视图（ConstraintLayout 本身）的左右两边，你可以添加两个约束。这两个约束就拉着 TextView 视图。

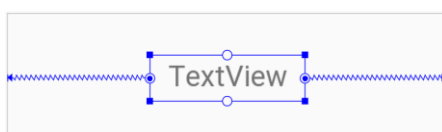


图 2-8 左右两边添加了约束的 TextView

你也可以创建四个方向上的约束（上、下、左和右）。如图 2-9 所示，这样，它们就可以均等地向上下和左右相反的方向拉，TextView 就会处于正中间的位置。



图 2-9 添加了四个约束的 TextView

因此，在 ConstraintLayout 里布置视图，只需要给它们添加上约束就可以了。

而要想控制组件的大小，可以在组件属性视图中调整 layout\_width 和 layout\_height，如图 2-10 所示。这里有三种选择：wrap\_content（组件自己根据内容决定），match\_parent（充满约束布局）和手动调整。



图 2-10 控制组件的大小

## 2.3.4 组件

有了上述组件的布置方法，只需要一个 `ConstraintLayout`，就可以布置多个布局。接下来，我们就看看如何使用约束布局布置 `activity_login` 布局。

### 1. 删除组件

目前，布局中的 `TextView` 组件不是我们想要的，需要首先删除。单击底部的“Design”标签页，如图 2-11 所示，选中 `TextView` 组件，单击右键，在弹出窗口中选择 `Delete`，此时 `TextView` 组件即被删除。

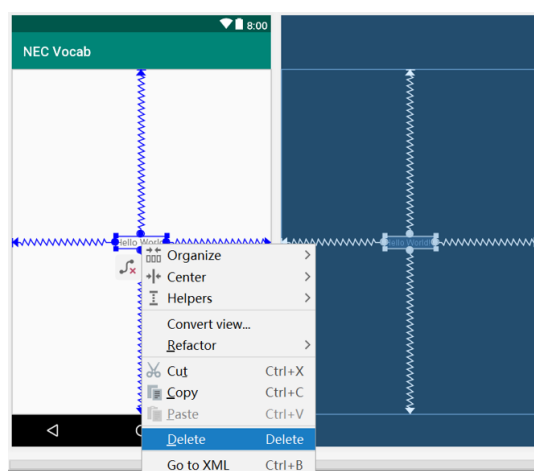


图 2-11 删除 `TextView` 组件

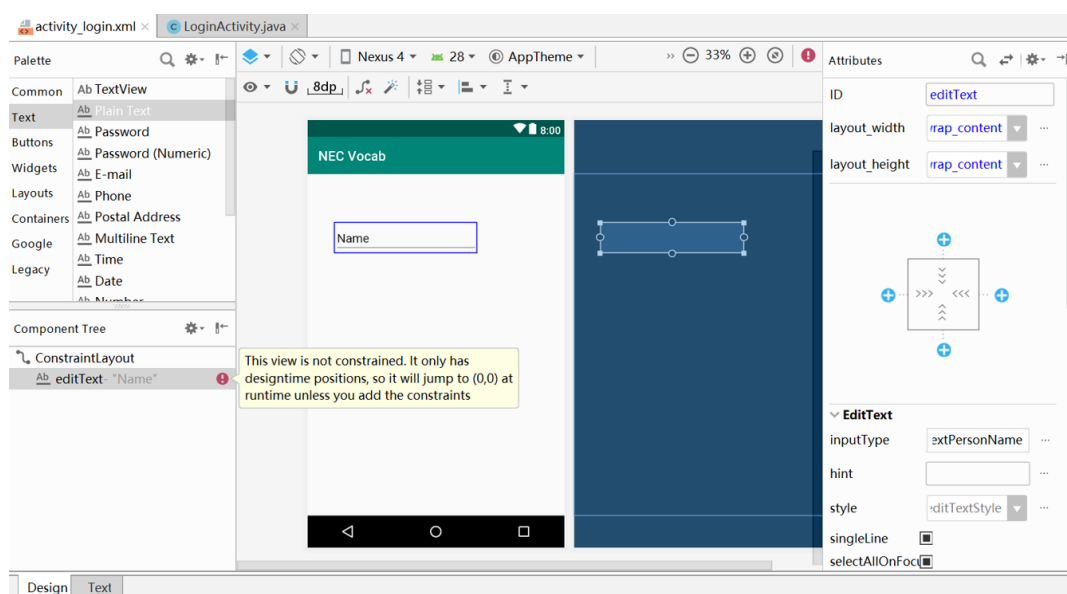


图 2-12 添加组件和没有添加约束的提示

## 2. 添加组件

在右上角的**组件面板**中选中 **Text→Plain Text**，将其拖拽到如图 2-12 所示的手机屏幕中。

在图中可以看到，出现了一个错误❗，将鼠标放在其上时会提示 **EditText** 没有添加约束。如果一个控件没有添加任何的约束，它在运行之后会自动位于界面的左上角。

## 3. 为组件添加约束

可以将约束添加到 **ConstraintLayout**，如本例中的第一个 **EditText** 约束就是相对于 **ConstraintLayout** 进行的定位（也可以将约束添加到另一个控件进行定位）。此时，如果位置调整好，可以直接点击**组件属性**视图（Attributes）栏下的约束图中的四个<sup>+</sup>，如图 2-13 左图所示。当然，在添加完约束之后，仍然可以调整约束位置，如图 2-13 右图所示。

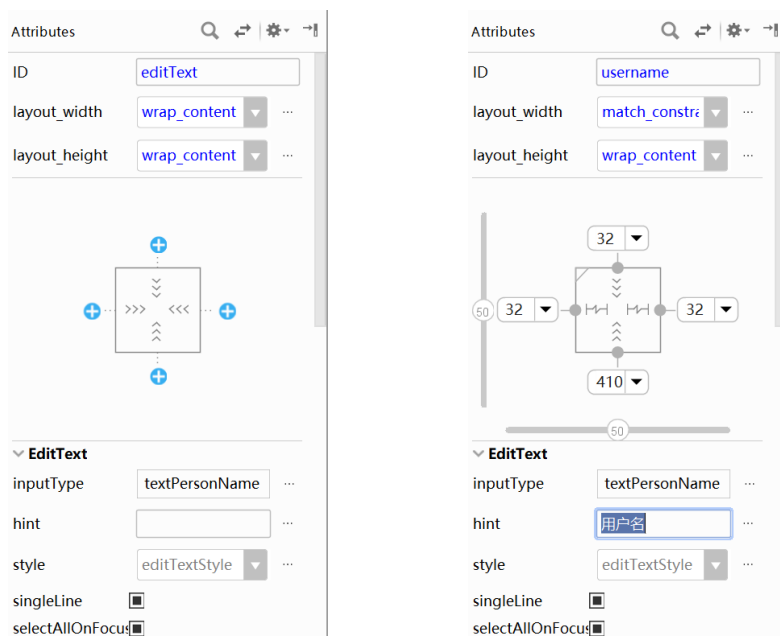


图 2-13 添加约束

在图 2-13 右图中，我们还对 **EditText** 组件的属性进行了设置，如将 **ID** 改为 **username**，**layout\_width** 设为 **match\_constraint**（注意，在代码中设置的是 **0dp**），将 **hint** 设为“用户名”等。

## 4. 添加其他相对组件

再添加一个用户输入密码的 **EditText**。在右上角的菜单栏中选择 **Text→Password**，让它位于第一个 **EditText** 的正下方，并通过都是圆圈之间的拖拽链接，建立约束，调整好位置。

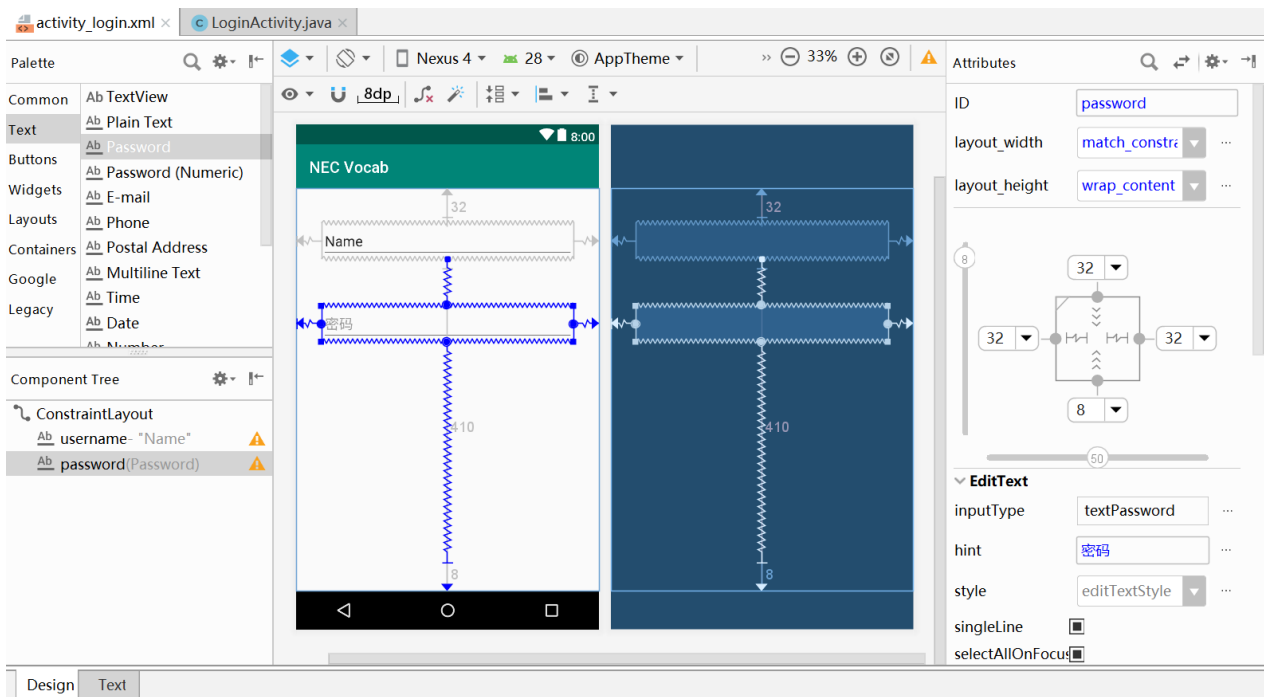


图 2-14 添加密码 EditText 组件

和用户名组件一样，将 ID 改为 password，**layout\_width** 设为 match\_constraint（在代码中设置的是 0dp），将 **hint** 设为“密码”等。

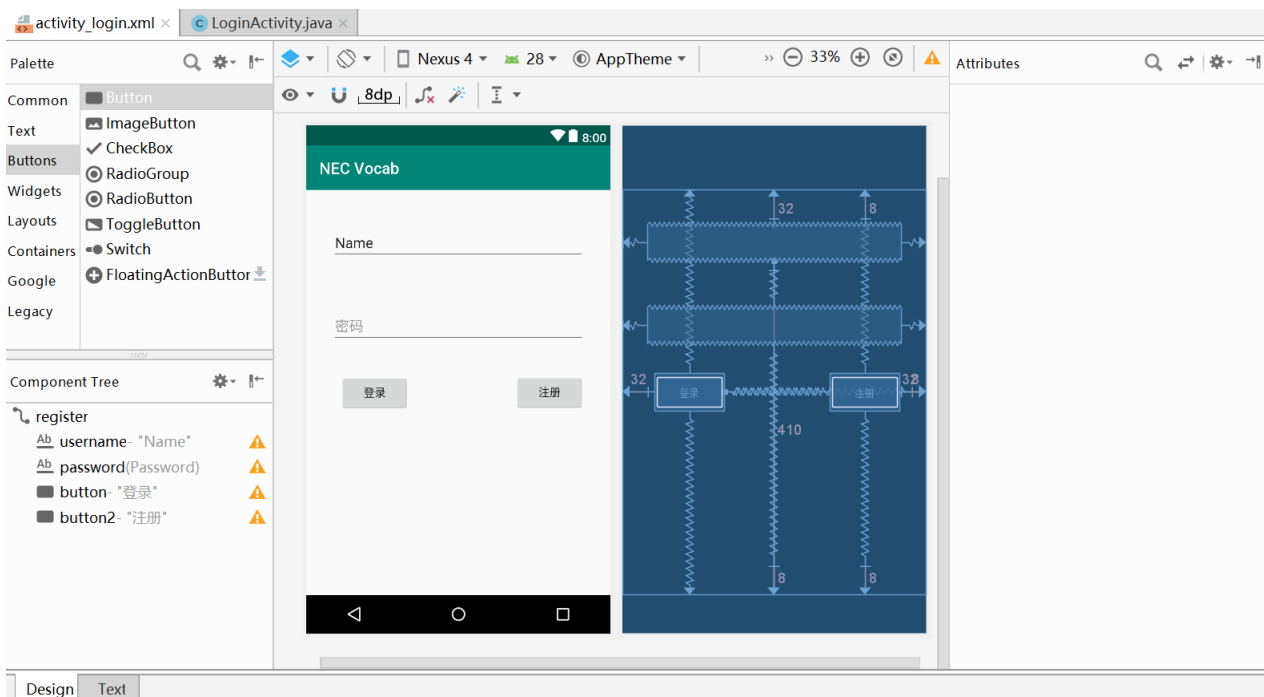


图 2-15 再添加两个 Button

最后，我们需要两个 Button，并将其设为“登录”和“注册”，以使用户点击此按钮时能进入

主界面或进入注册界面。

在右上角的菜单栏中选中 **Buttons→Button**，将其拖拽到如图 2-15 所示的手机屏幕中。让它们位于第二个 **EditText** 的下方，调整好位置，并通过都是圆圈之间的拖拽链接，建立约束。

和 **EditView** 组件一样，两 **Button** 组件的 **ID** 分别设置为 **login** 和 **register**，并将它们的 **text** 分别设置为“登录”和“注册”。

## 5. 删除约束

如果想删除约束，点击对应约束的圆圈就会删除对应的约束。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。

关于 **ConstraintLayout** 的先介绍这么多，对于本章及后面几章已经够用了。今后我们将在需要的时候再进行介绍。

接下来，我们看一下 **activity\_login.xml** 的参考代码清单 2-2（由于拖拽的位置和约束的固定点不一定相同，代码清单 2-2 仅供参考）。

### 代码清单 2-2 在 XML 文件中定义组件（**activity\_login.xml**）

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity">

    <EditText
        android:id="@+id/username"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="32dp"
        android:layout_marginBottom="410dp"
        android:ems="10"
        android:hint="用户名"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <EditText
        android:id="@+id/password"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```


        android:layout_marginStart="32dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="32dp"
        android:layout_marginBottom="8dp"
        android:ems="10"
        android:hint="密码"
        android:inputType="numberPassword"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/username"
        app:layout_constraintVertical_bias="0.105"/>

<Button
    android:id="@+id/login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="32dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:text="登录"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/register"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/password"
    app:layout_constraintVertical_bias="0.233"/>

<Button
    android:id="@+id/register"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="32dp"
    android:text="注册"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="1.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/password"
    app:layout_constraintVertical_bias="0.226"/>
</android.support.constraint.ConstraintLayout>

```

## 2.3.5 字符串资源

在图 2-15 中还有几个警告 ，将鼠标放于其上，可见提示，建议我们利用“@strings”存放“用户名”、“密码”、“登录”和“注册”等字符串资源，而不是将其“硬编码”在布局文件中。

EditText 与 Button 组件具有 **android:hint** 和 **android:text** 属性。这些属性指定组件显示的文字内容。目前我们以硬编码形式设置了这些组件的文本属性，如 **android:text="登录"**，但这通常不是个好方法。将文字内容放置在独立的字符串资源 XML 文件中，然后引用它们才是好方法。在后续章节中，我们将学习如何使用字符串资源轻松实现本地化。

每个项目都包含一个名为 **strings.xml** 的默认字符串文件。

在包浏览器中，找到 **res/values** 目录。点击小三角显示目录内容，然后打开 **strings.xml** 文件。添加应用布局需要的四个新的字符串，如代码清单 2-3 所示。

### 代码清单 2-3 添加字符串资源 (**strings.xml**)

```
<resources>
  <string name="app_name">NEC Vocab</string>
  <string name="username">用户名（手机号码）</string>
  <string name="password">输入密码</string>
  <string name="login">登录</string>
  <string name="register">注册</string>
</resources>
```

保存 **strings.xml** 文件。接下来，修改 **activity\_login.xml** 布局文件中的硬编码部分。

打开 **activity\_login.xml** 文件，单击底部的“**Design**”选项卡，选中 ID 为 **username** 的 **TextView** 组件，在**组件属性 (Attributes)** 栏下方，**hint** 的属性框中删除“用户名”，并点击旁边的“...”，在弹出的对话框中选择 **username**，点击“OK”，如图 2-16 所示。这样便完成了 **username** 组件字符串资源的添加。现在，NEC Vocab 项目的任何 XML 文件中，只要引用到 **@string/username**，应用运行时都会得到文本“用户名（手机号码）”。

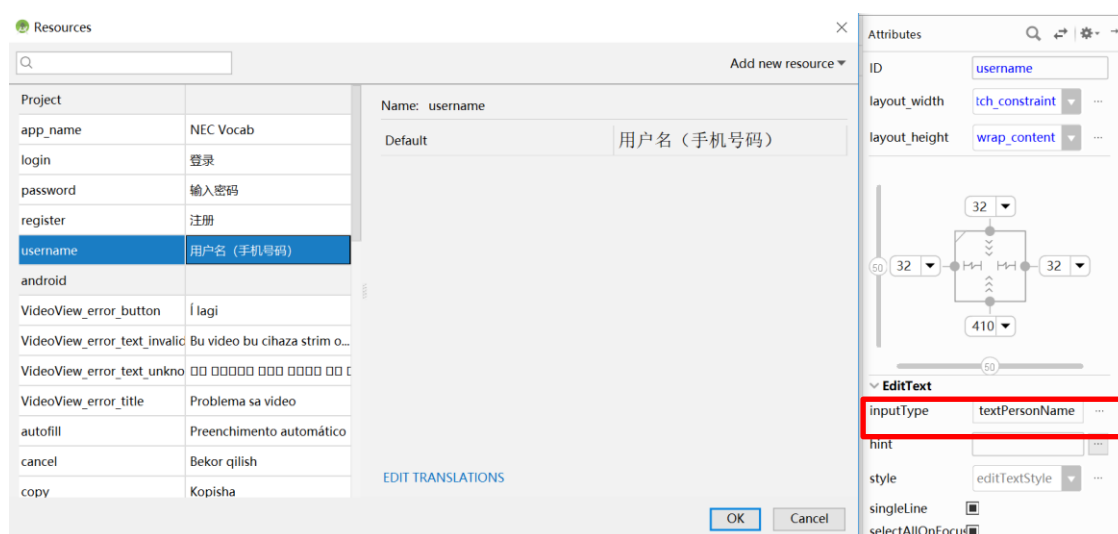



图 2-16 为 **username** 组件添加字符串资源



以同样的方式，可以为 password 组件、login 组件和 register 组件添加上字符串资源。

注意 android:id 属性值前面有一个+标志。而 android:text 属性值则没有。这个是因为我们将要创建资源 ID，而对字符串只是做了引用。

字符串文件默认命名为 strings.xml，当然也可以按个人喜好任意命名。一个项目可以有多个字符串文件。只要这些文件放置在 res/values/目录下，并且含有一个 resources 根元素，以及多个 string 子元素，字符串定义即可被应用找到并得到正确使用。

至此，应用的界面布局已经完成，现在我们使用模拟器或真机来运行程序。点击菜单栏中间的  按钮，在出现的对话框中选择适当的真机或模拟器，点击 OK。

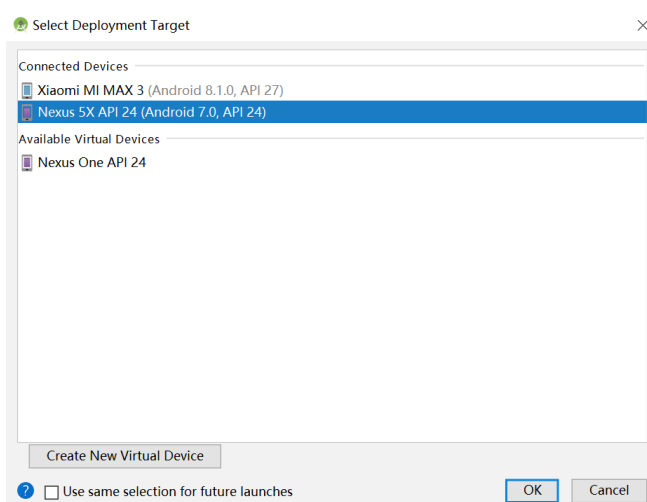


图 2-16 选择模拟器运行程序



图 2-17 NEC Vocab 应用的运行效果

可以看到如图 2-17 所示的屏幕。你可以输入用户名和密码。但点击登录或注册按钮，NEC Vocab 应用将不会有什么反应。应为我们还没有将布局（layout）与活动（activity）关联起来。

## 2.4 从布局 XML 到视图对象

想知道 activity\_login.xml 中的 XML 元素是如何转换为视图对象的吗？答案就在于 LoginActivity 类。

创建 NEC Vocab 项目的同时也创建了一个名为 LoginActivity 的 Activity 的子类。LoginActivity 类文件存放在 app/java 目录下。该 java 目录是项目全部 java 源代码的存放处。

在包浏览器中依次展开 app/java 目录与 com.studio.aime.necvocab 包，显示其中的内容。然后打开 LoginActivity.java 文件，其代码清单 2-4 所示，仔细查看其中的代码。

代码清单 2-4 LoginActivity.java 的默认代码（LoginActivity.java）

```
package com.studio.aime.necvocab;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class LoginActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
    }
}
```

（如果不能看到全部类包导入语句，请单击第一行导入语句左边的⊕符号，从而显示全部导入语句。）

该 java 类提供了一个 Activity 方法：onCreate(Bundle)。（这里 AppCompatActivity 是 Activity 类的一个子类，用于支持与较早 Android 版本的兼容性。）

Activity 子类的实例创建后，onCreate(Bundle)方法将会被调用。Activity 创建后，它需要获取并管理属于自己的用户界面。获取 activity 的用户界面，可以调用以下 Activity 方法：

```
public void setContentView(int layoutResID)
```

通过传入布局的资源 ID 参数，该方法生成指定布局的视图，并将其放置在屏幕上。布局视图生成后，布局文件包含的组件也随之以各自的属性定义完成实例化。

## 2.4.1 资源与资源 ID

布局是一种资源。资源是应用中非代码形式的内容，如 XML 文件、图像文件以及音频文件等。

项目的所有资源文件都存放在目录 `res/` 的子目录下。通过包浏览器可以看到，布局 `activity_login.xml` 资源文件存放在 `res/layout/` 目录下，包含字符串资源的 `strings.xml` 文件存放在 `res/values/` 目录下。

可以使用资源 ID 在代码中获取相应的资源。如 `activity_login.xml` 文件定义的布局资源 ID 为 `R.layout.activity_login`。

要查看 Situation English 的资源，展开 `app/build/generated/source/r/debug` 目录的内容。在这个目录中找到项目的包名，并在包中打开 `R.java` 文件。因为这个文件是在 Android 编译过程中自动生成的，正如是在该文件的顶部所警告的那样，不要改变它。

如果对资源进行了更改，可能无法立即看到该文件更新。针对生成的代码，Android Studio 维护一个隐藏的 `R.java` 文件。这里看到的 `R.java` 文件是为以前的应用程序生成的。当运行应用程序时，就会看到这个文件的更新。

### 代码清单 2-5 NEC Vocab 当前的资源 ID (R.java)

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package com.studio.aime.necvocab;

public final class R {
    public static final class anim {
        .....
    }
    .....
    public static final class layout {
        .....
        public static final int activity_login = 0x7f09001b;
        .....
    }
    .....
    public static final class string {
        .....
        public static final int app_name = 0x7f0b001d;
```

```

public static final int username=0x7f0b001e;
public static final int password=0x7f0b001f;

.....

public static final int login=0x7f0b0022;
public static final int register=0x7f0b0023;
}

.....

```

可以看到 `R.layout.activity_login` 即来自该文件。`activity_login` 是 `R` 的内部类 `layout` 里的一个整形常量名。

我们定义的字符串同样具有资源 ID。目前为止，我们还未在代码中引用过字符串，但如果需要，则应该使用以下方法：

```
setText (R.string.app_name);
```

`Android` 为整个布局文件以及各个字符串生成资源 ID，但 `activity_login.xml` 布局文件中的组件除外，因为不是所有的组件都需要资源 ID。在本讲中，我们要两个 `EditText` 组件和两个 `Button` 组件，因此需要为这四个按钮生成相应的资源 ID。

要为组件生成资源 ID，在定义组件时为其添加上 `android:id` 属性。在 `activity_login.xml` 文件中分别为两个按钮添加上 `android:id` 属性，如代码清单 2-2 中黑体部分所示。

注意 `android:id` 属性值前面有一个+标志。这个是因为我们将要创建资源 ID。

接下来我们来编码使用按钮组件，这需要以下两个步骤：

- 引用生成的视图对象；
- 为对象设置**监听器**，以响应用户操作。

## 2.4.2 组件的引用

`EditText` 和 `Button` 组件有了资源 ID，就可以在 `LoginActivity` 中直接获取它们。首先，在 `LoginActivity.java` 文件中增加两个成员变量。

在 `LoginActivity.java` 文件中输入代码清单 2-6 所示代码。

### 代码清单 2-6 添加成员变量（`LoginActivity.java`）

```

package com.studio.aime.necvocab;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
public class LoginActivity extends AppCompatActivity {

```

```

private EditText mUsername;
private EditText mPassword;
private Button mLogin;
private Button mRegister;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);
}
}

```

文件保存后，可能看到四个个错误提示。没关系，这点错误马上就可以搞定。请注意新增的四个成员（实例）变量名称的 `m` 前缀。该前缀是 Android 编程所遵循的命名约定。本课程将始终遵循该约定。

现在将鼠标移至代码的红色错误提示处。它们报告了相同的错误，如：Cannot resolve symbol ‘Button’。

这些错误告诉我们需要在 `LoginActivity.java` 文件中导入 `android.widget.Button` 类包。可以在文件的头部手动输入以下代码：

```
import android.widget.Button;
```

也可以让 Android Studio 的代劳。利用快捷键：**Alt+Enter** 自动导入单个包。类包导入后，刚才的错误提示就会消失了。当代码不正确时，此快捷方式通常是很有用的。经常试试！

如果错误仍然存在，检查 Java 代码以及 XML 文件，确认是否存在输入错误。

在 `activity` 中，可以通过以下 `Activity` 方法引用已经生成的组件：

```
public View findViewById(int id);
```

该方法接受组件的资源 ID 作为参数，返回一个视图对象。

在 `LoginActivity.java` 文件中，使用按钮的资源 ID 获取生成的对象后，赋值给对应的成员变量，如代码清单 2-7 所示。

#### 代码清单 2-7 引用组件（`LoginActivity.java`）

```

package com.studio.aime.necvocab;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
public class LoginActivity extends AppCompatActivity {
    private EditText mUsername;
    private EditText mPassword;
    private Button mLogin;
    private Button mRegister;
}

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);
    mUsername = findViewById(R.id.username);
    mPassword = findViewById(R.id.password);
    mLogin = findViewById(R.id.login);
    mRegister = findViewById(R.id.register);
}
}

```

### 2.4.3 监听器的设置

Android 应用属于典型的**事件驱动类型**。事件驱动型应用启动后即开始等待行为事件的发生，如用户单击某个按钮。（事件也可以由操作系统或其他应用触发。）

应用等待某个特定时间的发生，也可以说该应用正在“**监听**”特定事件。为响应某个事件而创建的对象叫做**监听器**（**listener**）。监听器是实现特定**监听器接口**的对象，用来监听某类事件的发生。

无需自己编写，Android SDK 已经为各种事件内置开发了很多监听器接口。当前应用需要监听用户的按钮“单击”事件，因此监听器需要实现 **View.OnClickListener** 接口。

首先处理“登录”和“注册”两个按钮。在 LoginActivity.java 文件中，在 onCreate(...)方法的变量赋值语句后输入下列代码，如代码清单 2-8 所示。

代码清单 2-8 为“下一词条”按钮设置监听器（LoginActivity.java）

```

package com.studio.aime.necvocab;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class LoginActivity extends AppCompatActivity {
    private EditText mUsername;
    private EditText mPassword;
    private Button mLogin;
    private Button mRegister;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        mUsername = findViewById(R.id.username);
        mPassword = findViewById(R.id.password);
        mLogin = findViewById(R.id.login);
    }
}

```

```

mRegister = findViewById(R.id.register);
mLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //目前什么都不做
    }
});

mRegister.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //目前什么都不做
    }
});
}
}

```

(如果遇到 **View cannot be resolved to a type** 类似的错误提示, 请使用 **Alt + Enter** 导入 **View** 类)。

在代码清单 2-8 中, 我们设置了两个监听器。当“登录” (mLogin) 或“注册” (mRegister) 按钮被点击后, 监听器会立即通知我们。setOnClickListener(OnClickListener)方法以监听器作为参数被调用。具体来说, 该方法以一个实现了 OnClickListener **接口**的对象作为参数调用。

## 1. 使用匿名内部类

传入 setOnClickListener(OnClickListener)方法的监听器参数是一个**匿名内部类** (anonymous inner class) 实现, 语法看上去稍显复杂。不过, 只需要记住最外层括号内的全部实现代码是作为整体参数传入 setOnClickListener(OnClickListener)方法内的即可。该传入的参数就是新建一个匿名内部类的实现代码。

```

mLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //目前什么都不做
    }
});

```


本课程所有的监听器都作为匿名内部类来实现。这样做有两个好处。第一, 因为匿名内部类的使用, 我们可以在同一处实现监听器方法, 代码更清晰可读; 第二, 事件监听器一般只在一处使用一次, 使用匿名内部类可避免不必要的命名类实现。

匿名内部类实现了 OnClickListener 接口, 因此它也必须实现该接口的唯一的 onClick(View)方法。onClick(View)方法的代码暂时是一个空结构。实现监听器接口需要实现 onClick(View)方

法，但具体如何实现由使用者来决定。因此，即使是空的实现方法，编译器也可以编译通过。

现在保存，然后看看这个应用的运行情况。

## 2.5 创建水平模式布局

运行 NEC Vocab 应用，然后旋转设备。（如果在模拟器上运行，点击  或使用 **Fn+Ctrl+F12/Ctrl+F12** 组合键）。

设备旋转后，NEC Vocab 应用看不见“用户名”了，如图 2-18 所示。这是怎么回事？

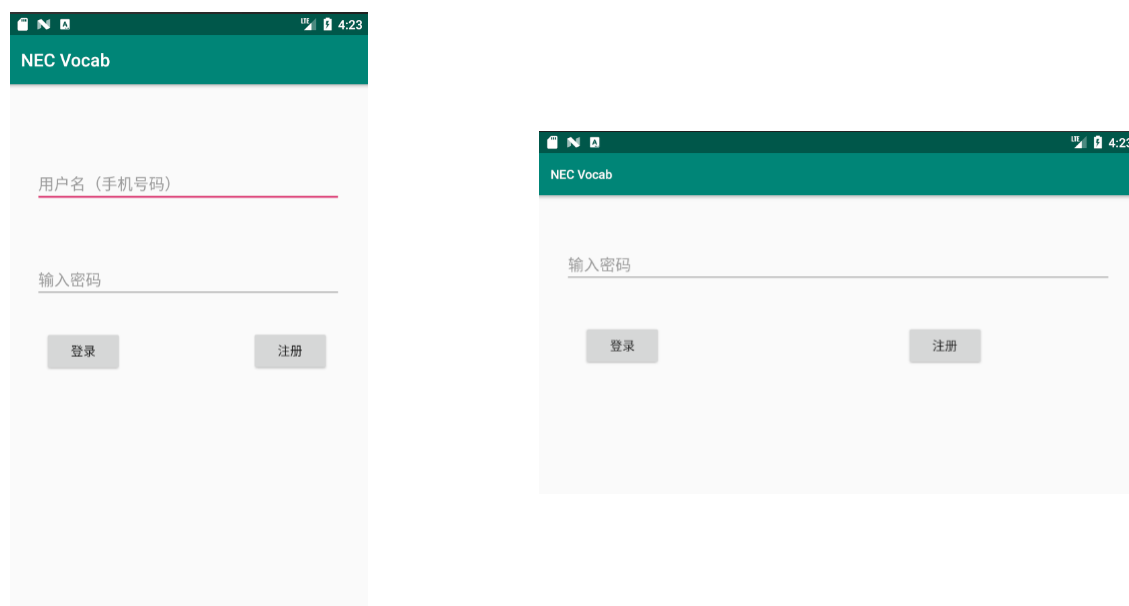


图 2-18 NEC Vocab 应用的旋转后运行效果，看不见“用户名”

### 2.5.1 设备配置与备选资源

旋转设备会改变了**设备配置**（**device configuration**）。设备配置是用来描述设备当前状态的一系列特征。这些特征包括：屏幕的方向、屏幕的密度、屏幕的尺寸、底座模式、键盘类型以及语言等。

通常，为匹配不同的设备配置，应用会提供不同的备选资源。为适应不同分辨率的屏幕，向项目里添加多套箭头图标就是这样一个使用案例。

设备的屏幕密度是一个固定的设备配置，无法在运行时发生改变。然而，有些特征，如屏



幕的方向，可以在应用运行时发生改变。

在**运行时配置变更**（runtime configuration change）发生时，可能会有更合适的资源来匹配新的设备配置。

下面我们研究一下 Android 如何匹配新的设备配置。我们新建一个备选资源，只要设备旋转至水平方向，Android 就会自动发现并使用它。

## 2.5.2 创建水平模式布局

在 Project 工具窗口右击 res 目录，并选择 **New→Android resource directory**。可以看到与图 2-19 类似的窗口，该窗口列出了资源类型以及这些资源类型的可用的限定符。在 **Resource type** 下拉框选择 **layout**。保留 **Source set** 选项仍为 **main**。接下来选择适合的布局资源。在 **Available quailifiers** (限定符) 栏目列表中选择 **Orientation** 并点击 >> 键将 **Orientation** 移到 **Chosen qualifiers** 栏目，如图 2-19 所示。

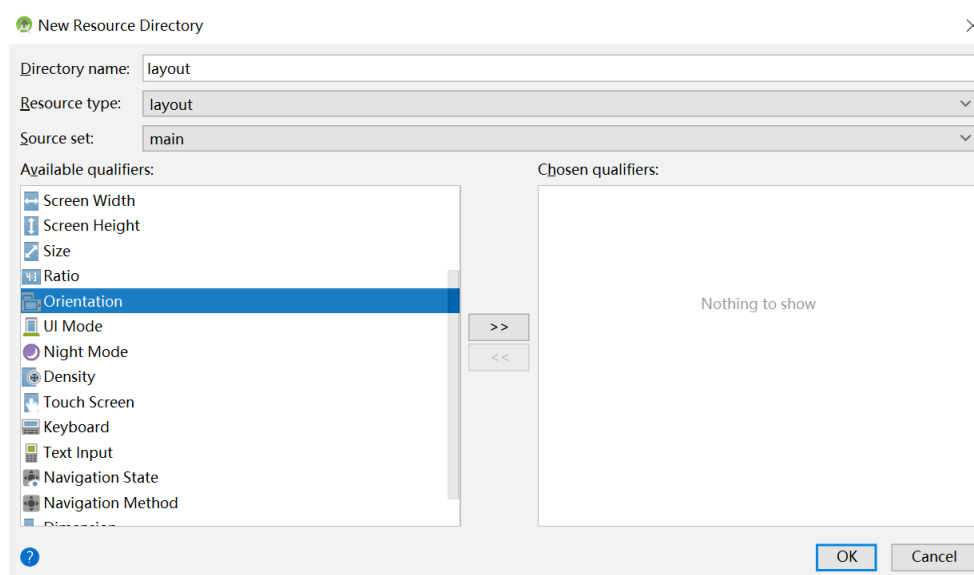


图 2-19 创建新的资源目录

最后，确保在 **Screen Orientation** 下拉列表中选 **Landscape**，如图 2-20 所示。验证目录名称为 **layout-land**。点击 **OK**，Android Studio 将创建 res/layout-land/文件夹。

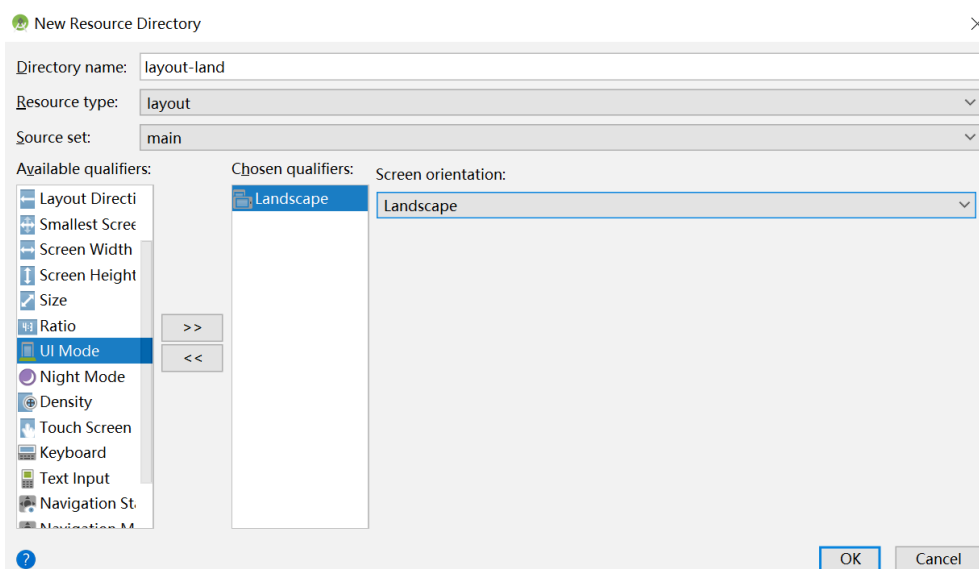


图 2-20 创建 res/layout-land 文件夹

这里的 **-land** 后缀名是配置修饰符的另一个使用例子。res 子目录的配置修饰符表明了 Android 是如何通过它来定位最佳资源以匹配当前设备配置的。

设备水平放置时，Android 会找到并使用 res/layout-land 目录下的布局资源。其他情况下，会默认使用 res/layout 目录下的布局资源。

然而，此时在 res/layout-land 目录下没有任何资源。

将 activity\_login.xml 文件从 res/layout/目录复制到 res/layout-land/目录中，并**保持文件名不变**。现在我们有了一个水平模式布局以及一个竖直模式的默认布局。注意，**两个布局文件必须具有相同的文件名**，这样它们才能以同一个资源 ID 被引用。

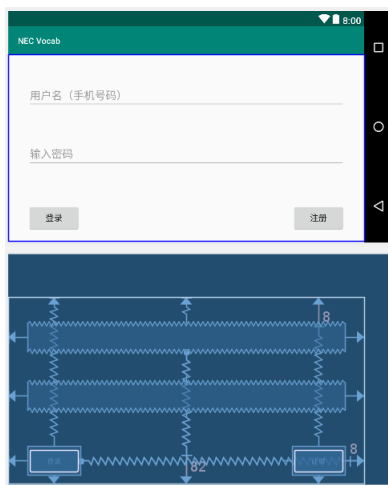


图 2-21 备选的水平模式布局

为了与默认的布局文件有所区别, 我们需要对水平模式布局文件做出一些修改。图 2-21 显示了将要对默认资源文件做出的修改 (通过拖拽, 删除和添加约束即可实现)。

再次运行 NEC Vocab 应用。旋转设备至水平位置。查看新的布局界面, 应当如图 2-21 所示。当然, **这里不仅仅是一个新的布局界面, 也是一个新的 VocabularyActivity。**

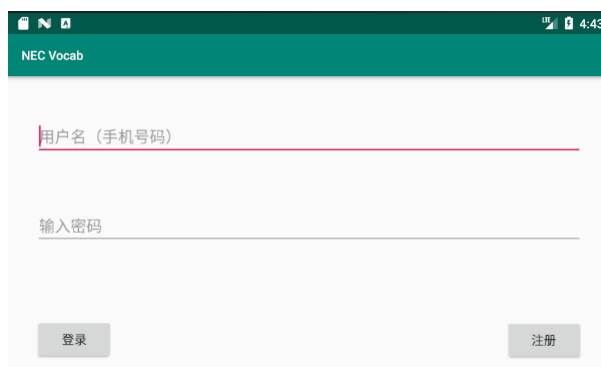


图 2-21 处于水平方位的 LoginActivity

设备旋转回竖直方位, 可以看到默认的布局界面以及另一个新的 LoginActivity。

Android 可以自动完成调用最佳匹配资源的工作, 但前提是它必须通过新建一个 activity 来实现。LoginActivity 要显示一个新布局, 必须再次调用 `setContentView(...)` 方法。而要调用 `setContentView(...)` 方法又必须先调用 `onCreate(...)` 方法。因此, 设备一经旋转, Android 需要销毁当前的 LoginActivity, 然后再新建一个 LoginActivity 来完成 `onCreate(...)` 方法的调用, 从而实现使用最佳资源匹配新的设备配置。

请记住, **只要在应用运行中设备配置发生了改变, Android 就会销毁当前的 activity, 然后再新建一个 activity。这是 Android 系统的 activity 规则。**

为了更清楚地看到这一过程, 我们来跟踪一下 Activity 的生命周期。

## 2.6 跟踪 Activity 的生命周期

本章最后我们介绍一下 Activity 的生命周期, 并通过日志跟踪生命周期。

每个 Activity 实例都有其生命周期。在生命周期内, activity 在**运行**、**暂停**和**停止**三种可能的状态间进行转换。每次状态发生转换时, 都有一个 Activity 方法将状态改变的消息通知给 activity。图 2-22 显示了 activity 的生命周期、状态以及状态切换时系统调用的方法。

用户可以与当前运行状态下的 activity 交互。设备上可以有很多应用，但是，**任何时候只能有一个 activity 处于用户能交互的运行状态。**

利用图 2-22 所示的方法，Activity 的子类可以在 activity 的生命周期状态发生关键性转换时完成某些工作。这些方法通常被称为生命周期回调方法。

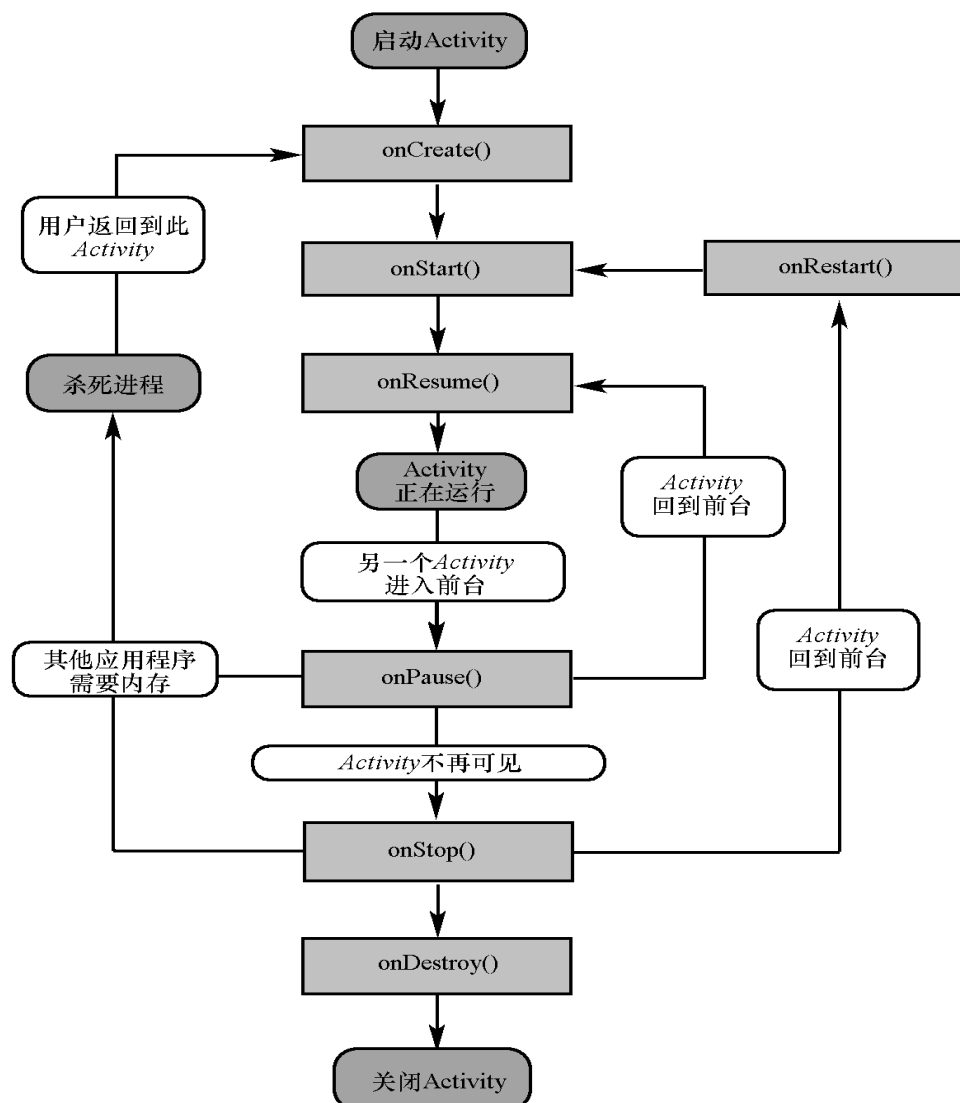


图 2-22 Activity 状态图解

我们已经熟悉了这些方法中的 `onCreate(Bundle)` 方法。在创建 activity 实例后，但在此实例出现在屏幕上之前，Android 操作系统会调用该方法。

通常，activity 通过覆盖 `onCreate(...)` 方法来准备以下用户界面相关的工作：

- 实例化组件并将组件放置在屏幕上（调用 `setContentView(int)` 方法）；
- 引用已经实例化的组件；
- 为组件设置监听器以处理用户交互；

- 访问外部模型数据。

**永远不要自己去调用 onCreate(...)方法或任何其他 Activity 生命周期方法。**我们只需要在 **Activity 子类里覆盖这些方法**即可。Android 会适时调用它们。

## 2.6.1 利用日志跟踪 Activity 生命周期

本节将通过覆盖 Activity 生命周期方法的方式介绍 LoginActivity 的生命周期。通过各个覆盖方法的日志输出，我们可以知道操作系统何时调用它们。

### 输出日志信息

Android 内部的 **android.util.Log** 类能够发送日志信息到系统级别的共享日志中心。Log 类有好几个日志信息记录方法。本讲义使用最多的是以下方法：

```
public static int d(String tag, String msg)
```

**d** 代表“**debug**”的意思，用来表示日志信息级别。其第一个参数表示信息的来源，第二个参数表示日志的具体内容。

该方法的第一个参数通常以类名为值的 **TAG** 常量输入。这样很容易看出日志的信息来源。

在 LoginActivity.java 中，为 LoginActivity 类新增一个 TAG 常量，如代码清单 2-9 所示。

#### 代码清单 2-9 新增一个 TAG 常量 (LoginActivity.java)

```
public class LoginActivity extends AppCompatActivity {
    private static final String TAG = "LoginActivity";
    .....
}
```

然后在 onCreate(...)方法里调用 **Log.d(...)**方法记录日志信息，如代码清单 2-10 所示。

#### 代码清单 2-10 为 onCreate(...)方法添加日志输出代码 (LoginActivity.java)

```
public class LoginActivity extends AppCompatActivity {
    private static final String TAG = "LoginActivity";
    .....

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "调用 onCreate(Bundle) 方法");
        setContentView(R.layout.activity_login);
        .....
    }
}
```

参照代码 2-10 输入相应的代码。事实上，在我们输入 **Log.d(...)** 方法的同时，Android Studio 已经自动导入了 **android.util.Log** 类，无需输入。接下来在 RegisterActivity 类中覆盖其他六个生命周期方法，如代码清单 2-11 所示。

#### 代码清单 2-11 覆盖更多生命周期方法（LoginActivity.java）

```
public class LoginActivity extends AppCompatActivity {
    private static final String TAG = "LoginActivity";
    .....

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "调用 onCreate(Bundle) 方法");
        setContentView(R.layout.activity_login);
        .....
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "调用 onStart() 方法");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "调用 onPause() 方法");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "调用 onResume() 方法");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "调用 onStop() 方法");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "调用 onDestroy() 方法");
    }

    @Override
    public void onRestart() {
        super.onRestart();
        Log.d(TAG, "调用 onRestart() 方法");
    }
}
```

注意，我们是先调用了超类的实现方法，然后再调用具体日志的记录方法。在 `onCreate(...)` 方法里，必须先调用超类的实现方法，然后再调用其它方法。而在其他方法中，是否首先调用超类方法就不重要了。

为何要使用 `@Override` 注解？可能一直以来都困惑着同学们。使用 `@Override` 注解，是要求编译器保证当前类具有准备覆盖的方法。例如，对于如下代码中名称拼写错误的方法，编译器将发出警告：

```
public class LoginActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        .....
    }
    .....
}
```

由于 `Activity` 类中不存在 `onCreat(Bundle)` 方法，因此编译器会发出警告。这样就可以改正拼写错误，而不是碰巧实现了一个名为 `LoginActivity.onCreate(Bundle)` 方法。

## 2.6.2 使用 LogCat

应用运行时，可以使用 LogCat 工具来查看日志。

当运行 NEC Vocab 时，在 Android Studio 底部选择 **Logcat** 标签，便出现如图 2-23 所示的 LogCat。

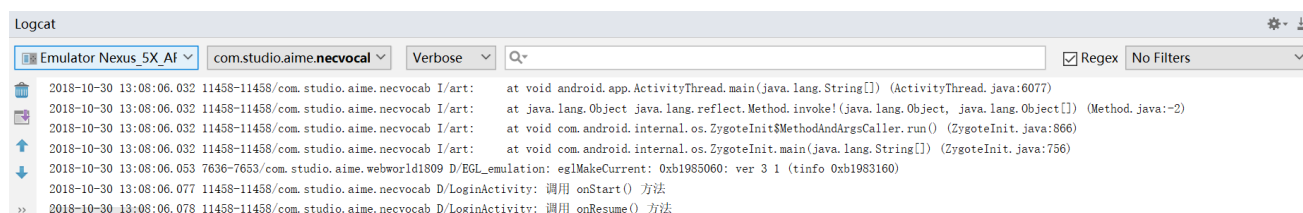


图 2-23 运行 LogCat 的 Android Studio

运行 NEC Vocab 时，消息将出现在现在 LogCat 中。默认情况下，显示生成的日志语句与应用程序的包名。用户会看到自己的信息以及一些系统输出信息。

要使用户的消息更容易找到，可以使用 TAG（标记）常量将其筛选输出。在 LogCat 中，点击位于 LogCat 右上角的筛选下拉菜单（见图 2-24），并选择 **Edit Filter Configuration**。



图 2-24 选择 Edit Filter Configuration

单击 LogCat 窗口左上方的**绿色+**按钮，创建一个消息过滤器。在 **Filter Name** 字段输入 LoginActivity，在 **Log Tag** 字段同样输入 LoginActivity，见图 2-25。

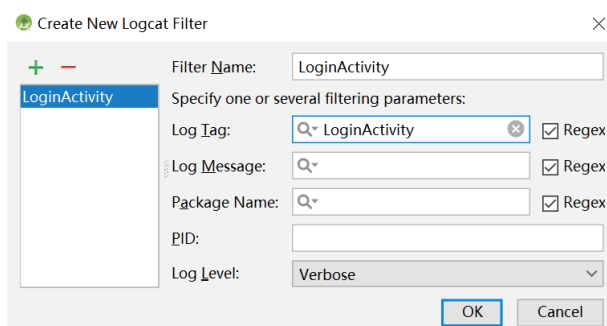


图 2-25 在 LogCat 中创建过滤器

点击 **OK**，在新出现的标签页窗口中仅显示了 Tag 为 LoginActivity 的日志信息(见图 2-26)。

我们看到：NEC Vocab 应用启动并完成 LoginActivity 初始实例的创建后，调用了三个生命周期方法。

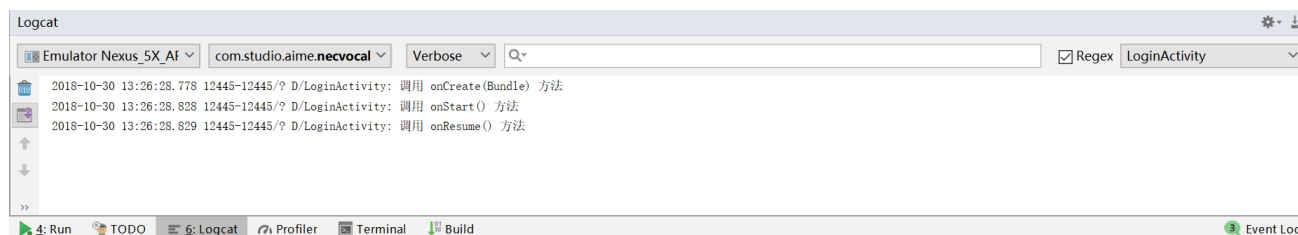


图 2-26 应用启动后被调用的三个生命周期方法

现在我们来做个实验。旋转设备，然后查看 LogCat。可以看到日志显示 LoginActivity 的 onPause()、onStop()和 onDestroy()方法被调用了，紧接着又调用了 onCreate(Bundle)、onStart()和 onResume()方法。如图 2-27 所示。

从日志我们看到：当设备旋转时，当前的 LoginActivity 实例会被销毁，然后创建一个新的 LoginActivity 实例。





图 2-27 设备选择日志记录

在设备上单击“后退”键，然后查看 LogCat。可以看到日志显示 LoginActivity 的 onPause()、onStop()和 onDestroy()方法被调用了，如图 2-28 所示。

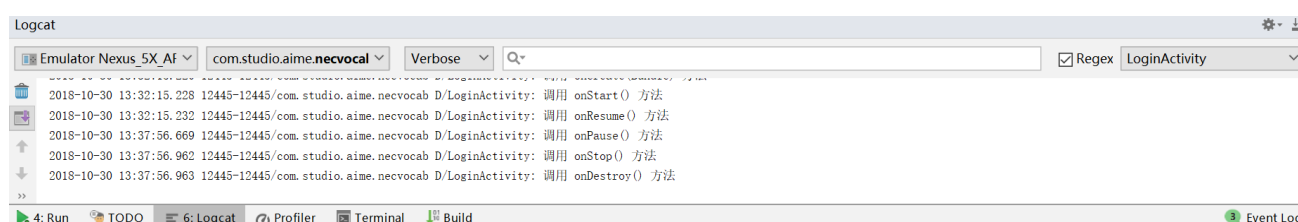


图 2-28 单击后退键销毁 activity

当按下设备的“后退”键时，相当于通知 Android：“我已经完成 activity，不再需要它了。”接到指令后，Android 系统立即销毁 activity。这实际是 Android 系统节约使用设备有限资源的一种方法。

重新运行 NCE Vocab 应用。这次，选择按下“主屏幕”键，然后查看 LogCat。日志显示系统调用了 onPause()和 onStop()方法，但并没有调用 Destroy() 方法，如图 2-29 所示。

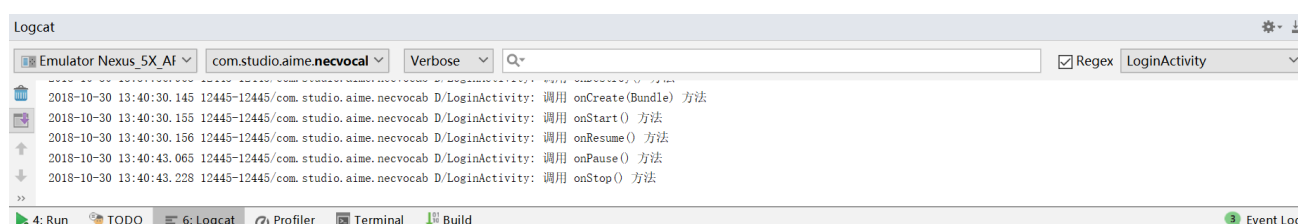


图 2-29 按下主屏幕键停止 activity

按“主屏幕”键，相当于通知 Android：“我去别处看看，稍后可能回来。”此时，为快速响应随时返回应用，Android 知识暂停当前的 activity，并不销毁它。

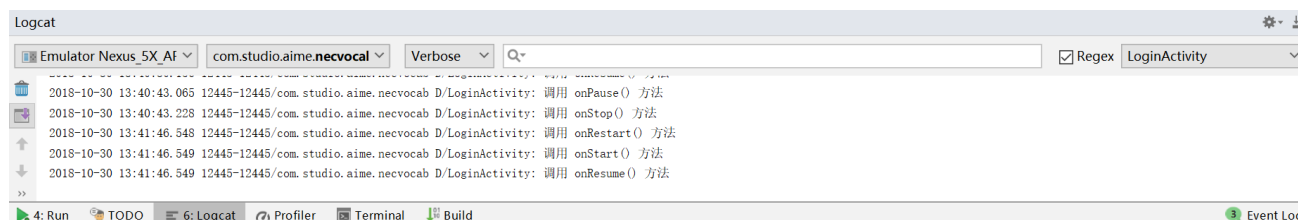


图 3-30 再次启动 NCE Vocab 应用

再次启动 NCE Vocab 应用, 日志显示系统调用了 `onRestart()`、`onStart()`和 `onResume()`方法。

需要注意的是, 谁也无法保证停止的 `activity` 能够存在多久。比如系统需要回收内存, 它首先要销毁那些停止的 `activity`。

最后, 想象一个会部分遮挡住当前 `activity` 界面的小的弹出窗口 (如小广告)。它出现时, 被遮住 `activity` 会被系统暂停, 用户也无法与它交互。当弹出小窗口关闭时, 被遮住的 `activity` 将会重新开始运行。

在本课程学习过程中, 为完成各种实际任务, 需要覆盖不同的生命周期方法。