

2018

Android 移动开发实训

程源

广东机电职业技术学院

2018/9/1

目录

4.1 本章目标	- 2 -
4.2 创建带有底部导航的界面	- 3 -
4.3 添加菜单项	- 6 -
4.2.1 菜单项	- 6 -
4.2.2 添加 Android vector 图标	- 7 -
4.2.3 修改菜单项并添加条目	- 9 -
4.4 使用菜单选项	- 11 -
4.5 设备旋转与 Activity 生命周期	- 13 -
4.5.1 设备配置与备选资源	- 14 -
4.5.2 设备旋转前保存数据	- 16 -
4.5.3 再探 Activity 生命周期	- 19 -
4.6 Android 应用的调试.....	- 20 -
4.6.1 异常与栈跟踪	- 21 -
4.6.2 记录栈跟踪日志	- 23 -
4.6.3 设置断点	- 24 -

4 创建底部导航界面

4.1 本章目标

在第二、第三章中我们实现了 NEC Vocab 应用的注册登录模块，并将用户的注册信息存放在 SharedPreferences 中。然而，当用户注册完成后，点击登录按钮时，NEC Vocab 应用除了给出“注册成功”的消息提示之外，并没有更进一步的动作。用户当然希望点击登录之后即可进入 NEC Vocab 应用的主界面，同时希望主界面能有一个底部导航栏，如图 4-1 所示。

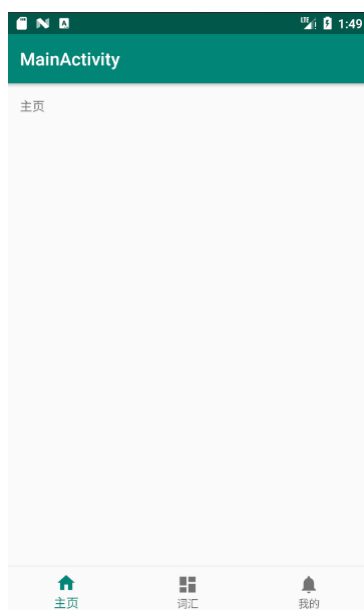


图 4-1 带有底部导航栏的主界面

现在主流的应用，基本的架构都是使用底部导航栏将内容分类，在不同选项，即 Tab（标签），间做切换内容。

可以通过自定义的方式来实现底部导航栏，通常每个 Tab 的 item 由一个 icon 和 title 组成。当然也可以使用官方在 support 包内提供的 BottomNavigationView 来实现。

本章我们将介绍如何利用 BottomNavigationView 实现带有底部导航栏的主界面，同时还将讲解如何处理设备旋转和应用中的 bug。

4.2 创建带有底部导航的界面

我们在利用向导创建 LoginActivity 和 RegisterActivity 时都选择了 Android Studio 提供的 Empty Activity。事实上，Android Studio 还提供了很多有预设功能的 Activity，Bottom Navigation Activity 便是其中之一。我们可以利用 Bottom Navigation Activity 创建带有底部导航栏的 Activity。

如图 4-2 所示，右击包 com.studio.aime.necvocab，选择 New → Activity → Bottom Navigation Activity。

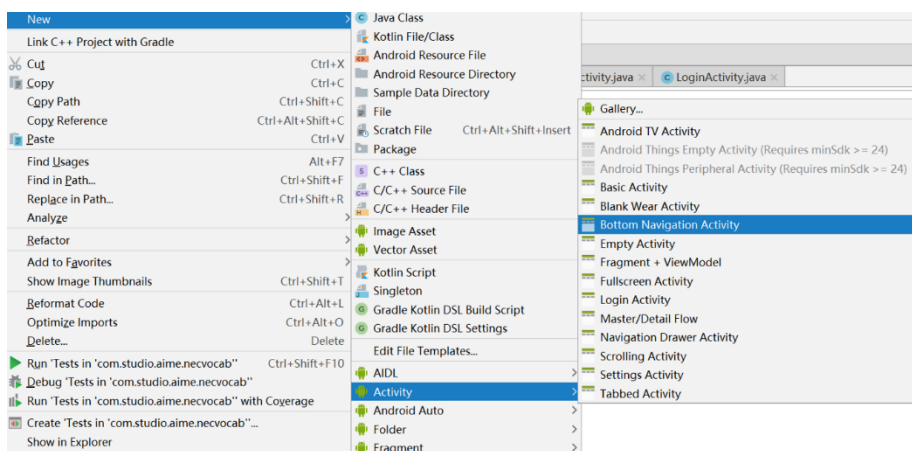


图 4-2 创建 Bottom Navigation Activity

在弹出的 Activity 配置对话框中，我们将保持默认配置，因为这个带底部导航栏的 Activity 将是用户登录后进入的主界面。当然，你也可以将 Activity Name 设为其他名称。

为了减少后续开发出现的错误，我们先运行一下 Nec Vocab 应用。为此，我们先修改 LoginActivity，让其“登录”按钮能通过 Intent 启动 MainActivity，如代码清单 4-1 所示。

代码清单 4-1 从 LoginActivity 启动 MainActivity（LoginActivity.java）

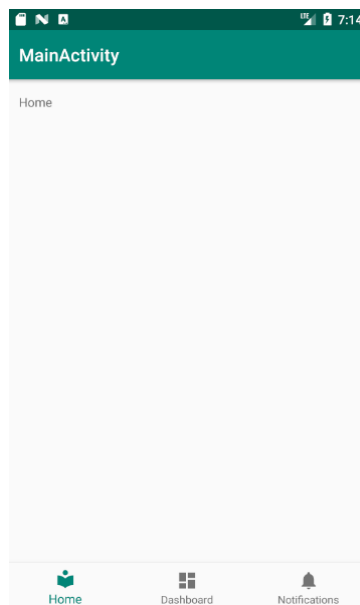
```
public class LoginActivity extends AppCompatActivity {
    .....
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        .....
        mLogin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if ((mUsername.getText().toString().trim()).equals(
                    mPreferences.getString("username", "")))
                    && (mPassword.getText().toString().trim()).equals(
```

```

        mPreferences.getString("password", "")))
    {
        Toast.makeText(LoginActivity.this,
            "登录成功", Toast.LENGTH_SHORT).show();
        Intent intent = new Intent(LoginActivity.this, MainActivity.class);
        startActivity(intent);
    } else {
        Toast.makeText(LoginActivity.this,
            "用户名或密码不正确", Toast.LENGTH_SHORT).show();
    }
}
});
.....
}
}

```

现在运行 NEC Vocab 应用，在登录界面输入正确的用户名和密码之后，点击登录，即进入如图 4-3 所示的带底部导航栏的主界面。



如图 4-3 带底部导航栏的主界面

现在，我们看看 Android Studio 的 Activity 向导做了什么。

正如所料，向导创建了一个名为 MainActivity 的 Java 文件，一个名为 activity_main.xml 的布局文件，并在 AndroidManifest.xml 配置文件中对 MainActivity 进行了注册：

```

<activity
    android:name=".MainActivity"
    android:label="@string/title_activity_main">
</activity>

```

在编辑区的 activity_main.xml 选项卡的图形布局工具栏中，我们可以看到如图 4-4 所示的

新建的带底部导航栏的主界面样式。

此外，它在 `app\build.gradle`（有关 `build.gradle` 我们将在今后的章节中介绍）中导入了 `support:design` 库：

```
implementation 'com.android.support:design:28.0.0'
```

因为 `BottomNavigationView` 就在这个 `design` 库中。而 `BottomNavigationView` 就是实现 Android 底部导航栏的主角。在 Google 在其 `Android Support Library 25` 及其以后的版本中增加了 `BottomNavigationView` 控件，并添加了创建 Activity 的 `Bottom Navigation Activity` 向导。

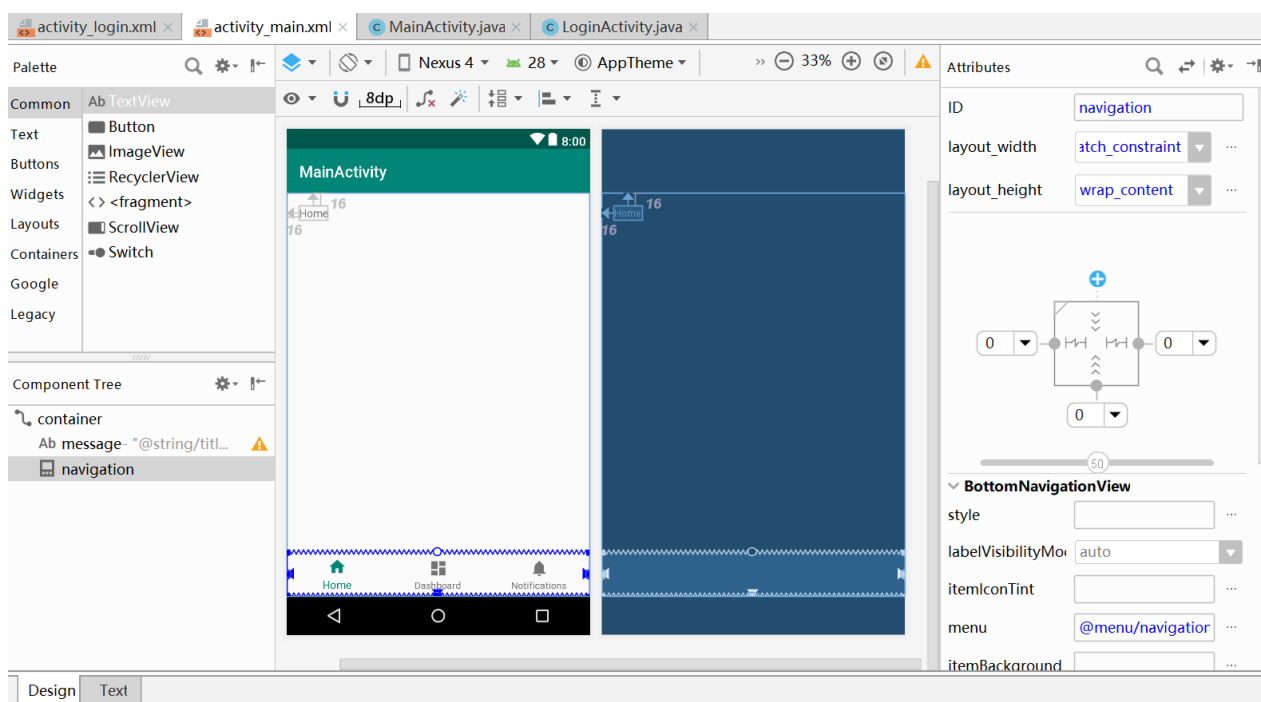


图 4-4 带底部导航栏的主界面

同时，向导还在 `res/menu` 目录下创建了一个名为 `navigation.xml` 文件的**菜单项**。

最后，向导还在字符串资源文件 `strings` 中添加了如下字符串资源：

```
<resources>
.....
<string name="title_activity_main">MainActivity</string>
<string name="title_home">Home</string>
<string name="title_dashboard">Dashboard</string>
<string name="title_notifications">Notifications</string>
</resources>
```

下面我们就来看看这些新创建的组件是如何协调工作的。

4.3 添加菜单项

我们首先从菜单项开始。

4.2.1 菜单项

底部导航栏由菜单项组成。打开 `res/menu/navigation.xml` 文件，选择 **Design** 标签页，可以看到如图 4-5 所示的页面。

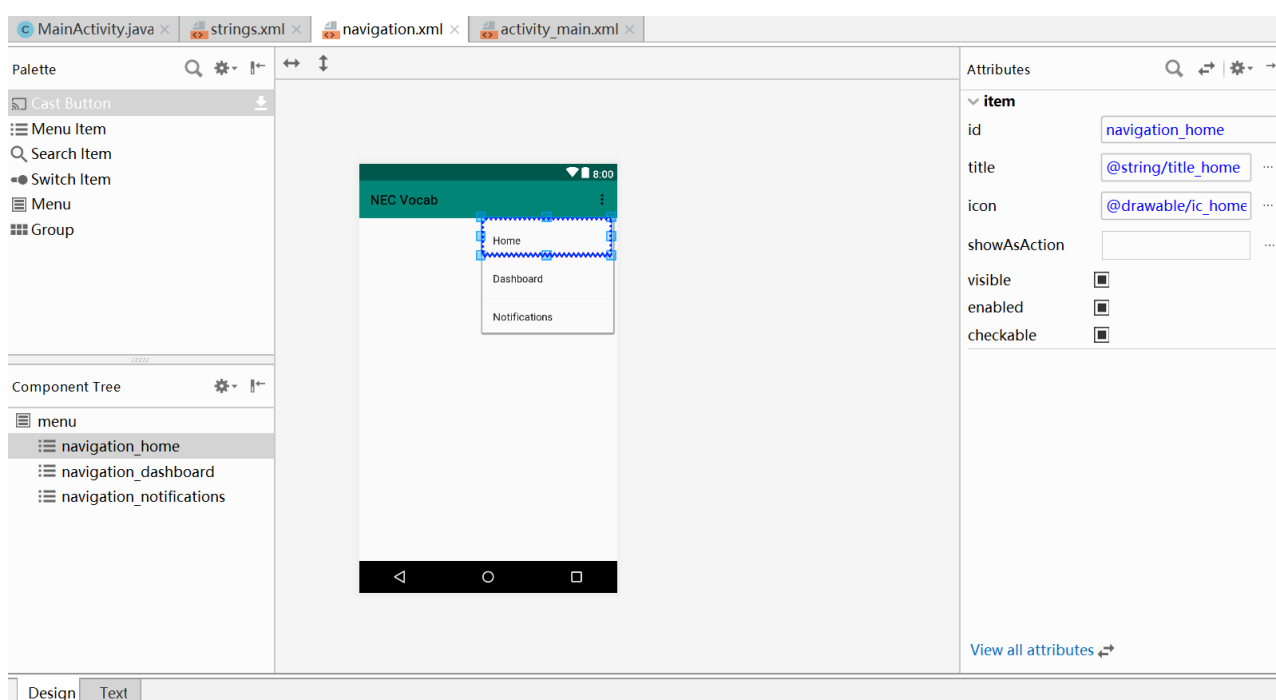


图 4-5 navigation 的页面

通过 **Text** 标签页可以看到 `navigation.xml` 文件的 xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/navigation_home"
        android:icon="@drawable/ic_home_black_24dp"
        android:title="@string/title_home"/>
    <item
        android:id="@+id/navigation_dashboard"
        android:icon="@drawable/ic_dashboard_black_24dp"
        android:title="@string/title_dashboard"/>
    <item
        android:id="@+id/navigation_notifications"
        android:icon="@drawable/ic_notifications_black_24dp"
```

```
        android:title="@string/title_notifications"/>
    </menu>
```

从中可以看到, 目前的 navigation 菜单有 3 个条目 (item), 其 id 分别是: navigation_home、navigation_dashboard 和 navigation_notifications。其图标 (icon) 具有 $\times\times\times_24dp$ 的形式, 即 vector 标签。这就是我们在图 4-4 中看到的底部的三个图标的形式。

当然, 向导自动创建的菜单选项并不能满足我们的需求, 下面我们对菜单项进行一些修改。

首先, 我们需要为菜单选项添加一系列字符串资源, 因为向导添加的字符串资源并非我们所需要的。参照代码清单 4-3, 添加并修改一些字符串资源。

代码清单 4-3 添加字符串资源 (strings.xml)

```
<resources>
    .....
    <string name="title_home">主页</string>
    <string name="title_voclib">词汇</string>
    <string name="title_discover">发现</string>
    <string name="title_mine">我的</string>
</resources>
```

下面我们来添加几个 vector 图标。

4.2.2 添加 Android vector 图标

vector 图标是一种矢量格式图, 在 xml 文件中的标签是 **<vector>**, 画出的图形可以像一般的图片资源使用, 例子如下:

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24.0"
    android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M12,11.55C9.64,9.35 6.48,8 3.8v11c3.48,0 6.64,1.35 9,3.55 2.36,-2.19
5.52,-3.55 9,-3.55V8c-3.48,0 -6.64,1.35 -9,3.55z"
    </path>
</vector>
```

这段代码所画出的图形为



图 4-6 典型的 vector 图标

Android Studio 提供了丰富的 vector 图标资源，我们可以直接使用（需要指出的是 5.0 以下的设备不支持）。在 Project 视图下，选中 res/drawable，右键选择 new → Vector Asset，将弹出如图 4-7 所示的对话框：

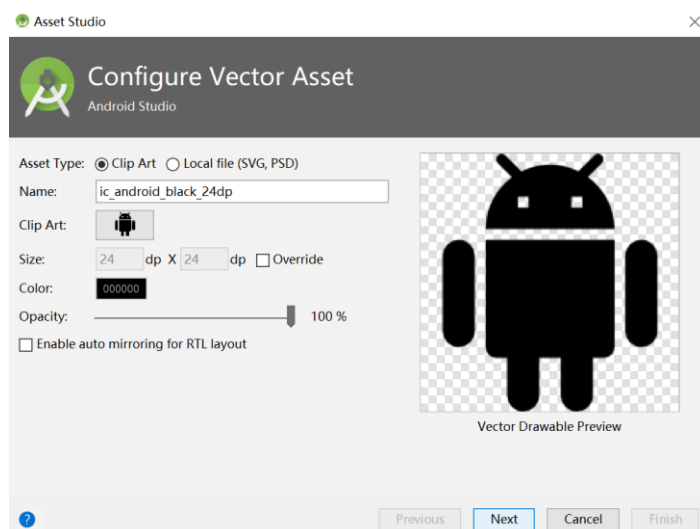


图 4-7 配置 vector 图标

点击 Clip Art:旁边得到“小机器人”，在弹出的如图 4-8 所示的选择图标窗口中选取你心仪的图标，例如本书所选的 local library 图标，点击 OK。

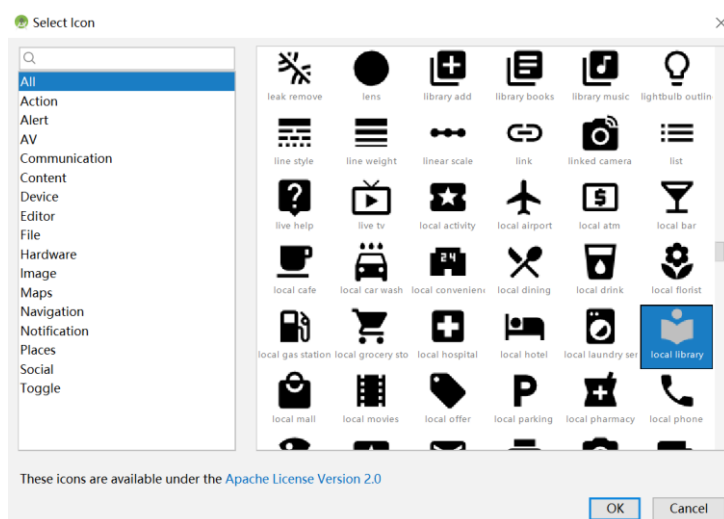


图 4-8 选择图标

在随后出现的窗口中，可以修改 Name:框中的默认图标名称。本例中我们将该图标命名为 ic_voc_library_black_24dp，表明是用于“词汇”标签选项的图标。用同样的方法，还可以添加其他 vector 图标，例如我们还添加了 ic_discover_black_24dp 和 ic_person_black_24dp 分别作为

“发现”和“我的”标签选项图标。

4.2.3 修改菜单项并添加条目

现在我们已经拥有了必要的图标资源，可以修改菜单项并添加条目了。

打开 navigation.xml 文件，选择 Design 标签页，从组件面板选中 Menu Item，并将其拖拽到预览界面中的 menu 框的下方，如图 4-9 所示。

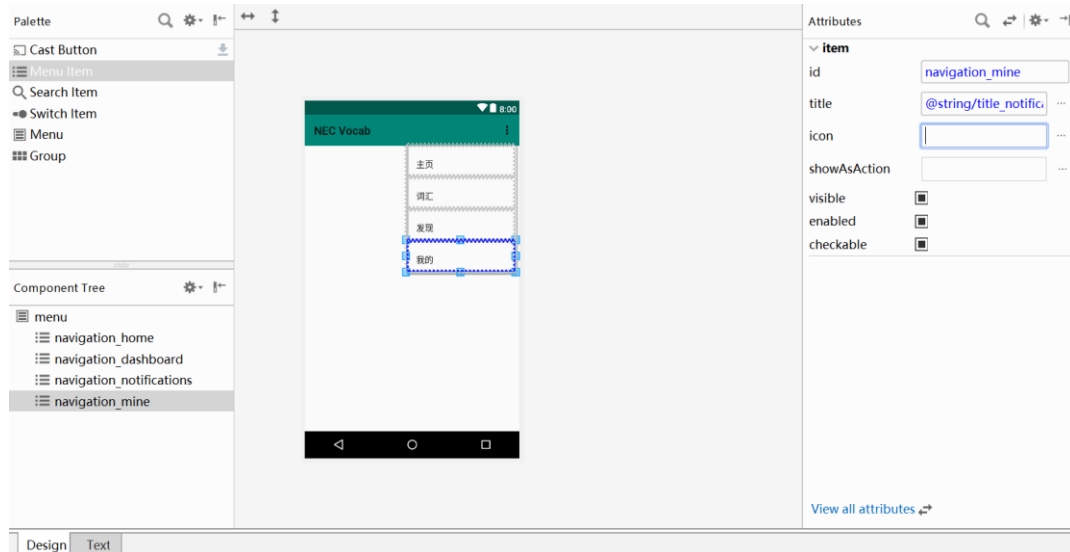


图 4-9 添加菜单栏的条目

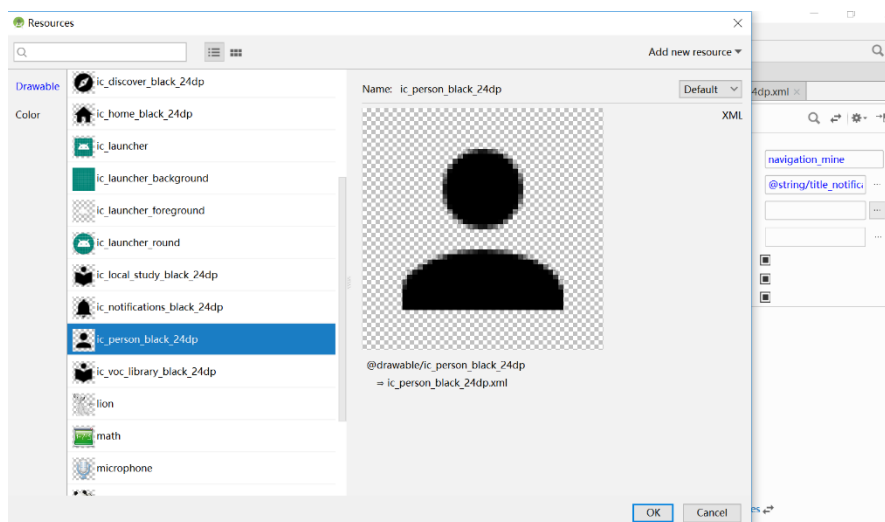


图 4-10 为菜单条目添加图标

接着，我们为 Item 添加图标资源。如图 4-10 所示，点击**属性栏**中的 icon 右边的 “...” 选中，在弹出的对话框中选择所需要的图标，点击 OK。

最后，修改 navigation.xml 文件中条目（Item）的 id 和 title，使其满足我们的要求。为了便于读者参考，代码清单 4-3 列出了 navigation.xml 文件的代码。

代码清单 4-3 菜单选项的 xml 代码（navigation.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/navigation_home"
    android:icon="@drawable/ic_home_black_24dp"
    android:title="@string/title_home"/>
  <item
    android:id="@+id/navigation_voclib"
    android:icon="@drawable/ic_voc_library_black_24dp"
    android:title="@string/title_voclib"/>
  <item
    android:id="@+id/navigation_discover"
    android:icon="@drawable/ic_discover_black_24dp"
    android:title="@string/title_discover"/>
  <item
    android:id="@+id/navigation_mine"
    android:icon="@drawable/ic_person_black_24dp"
    android:title="@string/title_mine"/>
</menu>
```

现在，我们看一下 activity_main.xml 文件中的**预览**，应该具有如图 4-11 所示的形式。

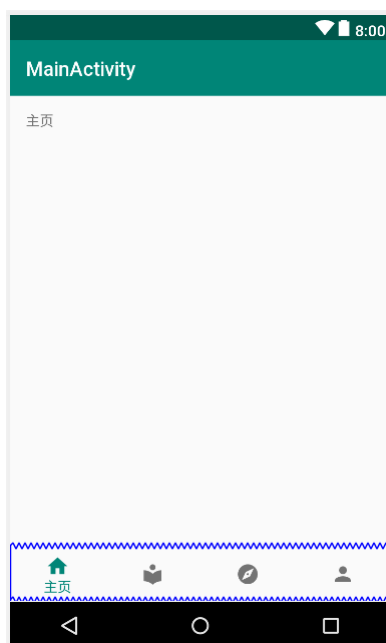


图 4-11 主界面预览图

4.4 使用菜单选项

现在我们还不能运行 NEC Vocab 应用。我们还需要对 MainActivity 做一些修改。

在此之前我们先考察一下布局文件 activity_main.xml 文件。为了显示的更清楚，我们修改了 TextView 组件字体的大小和颜色，同时取消了 text 的设置，而将 hint 设为“主界面”，如图 4-12 所示（注意右侧属性栏的修改）。

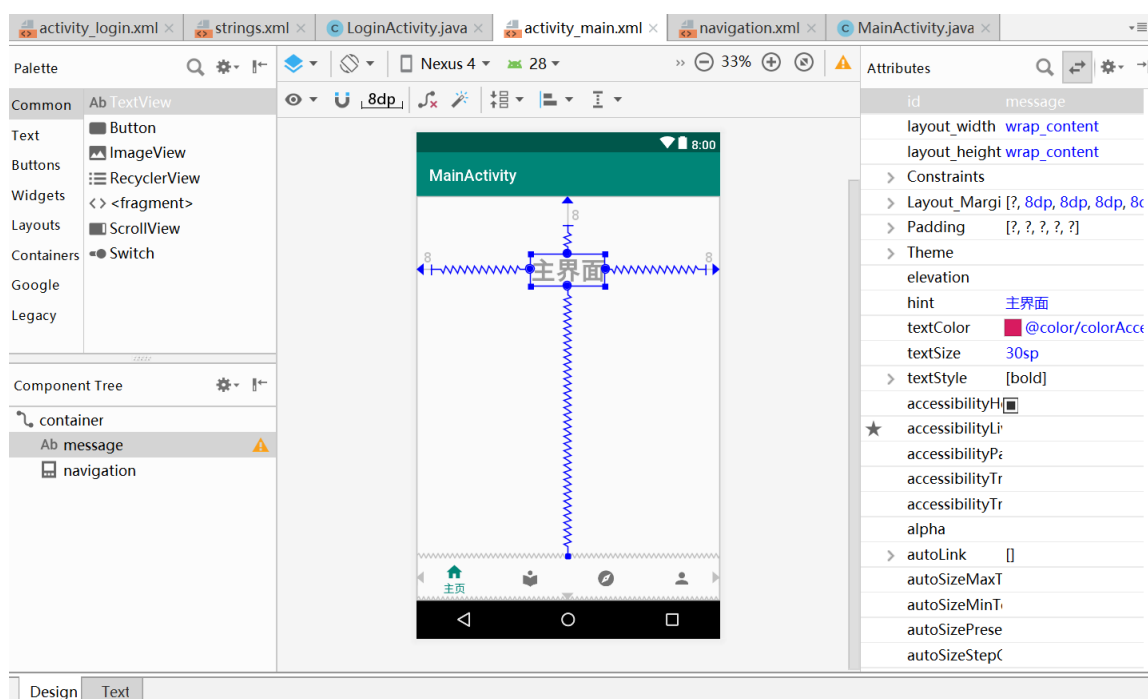


图 4-12 修改后的主界面布局

为了便于理解，代码清单 4-4 同时也列出了 activity_main.xml 的全部代码（注意粗体部分）。

代码清单 4-4 向导创建的布局文件（activity_main.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:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```

        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:hint="主界面"
        android:textColor="@color/colorAccent"
        android:textSize="30sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/navigation"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.501"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.163"/>
<android.support.design.widget.BottomNavigationView
    android:id="@+id/navigation"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="0dp"
    android:layout_marginEnd="0dp"
    android:background="?android:attr/windowBackground"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:menu="@menu/navigation"/>
</android.support.constraint.ConstraintLayout>

```

可以看到，向导使用了一个名为 BottomNavigationView 的组件。在该组件中引用了前面创建的菜单栏：app:menu="@menu/navigation"。

理解了布局文件，我们再来打开向导创建的 MainActivity 文件。当然我们需要对其进行一定的修改，包括 switch 语句中的内容和设置 TextView 的默认显示内容，如代码清单 4-6 所示。

代码清单 4-6 修改后的 MainActivity (MainActivity.java)

```

public class MainActivity extends AppCompatActivity {
    private TextView mTextMessage;
    private BottomNavigationView.OnNavigationItemSelectedListener mOnNavigationItemSelectedListener
        = new BottomNavigationView.OnNavigationItemSelectedListener() {
        @Override
        public boolean onNavigationItemSelected(@NonNull MenuItem item) {
            switch (item.getItemId()) {
                case R.id.navigation_home:
                    mTextMessage.setText(R.string.title_home);
                    return true;
                case R.id.navigation_voclib:
                    mTextMessage.setText(R.string.title_voclib);
                    return true;
                case R.id.navigation_discover:
                    mTextMessage.setText(R.string.title_discover);
                    return true;
                case R.id.navigation_mine:
                    mTextMessage.setText(R.string.title_mine);

```

```

        return true;
    }
    return false;
}
};
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mTextMessage = (TextView) findViewById(R.id.message);
    mTextMessage.setText(R.string.title_home);
    BottomNavigationView navigation = (BottomNavigationView) findViewById(R.id.navigation);
    navigation.setOnNavigationItemSelectedListener(mOnNavigationItemSelectedListener);
}
}

```

代码清单 4-6 中，首先定义了一个 `BottomNavigationView.OnNavigationItemSelectedListener` 类型的成员变量 `mOnNavigationItemSelectedListener`。该变量通过覆盖 `onNavigationItemSelectedListener` 方法，在 `OnNavigationItemSelectedListener` 内实现页面的切换。并在 `onCreate` 方法中，给 `BottomNavigationView` 的对象 `navigation` 设置一个 `OnNavigationItemSelectedListener` 用来响应切换的动作。在该方法里我们还设置了默认显示内容 `mTextMessage.setText(R.string.title_home)`

现在运行 NEC Vocab 应用，并切换底部导航标签，应该可以看到如图 4-12 所示的主界面。

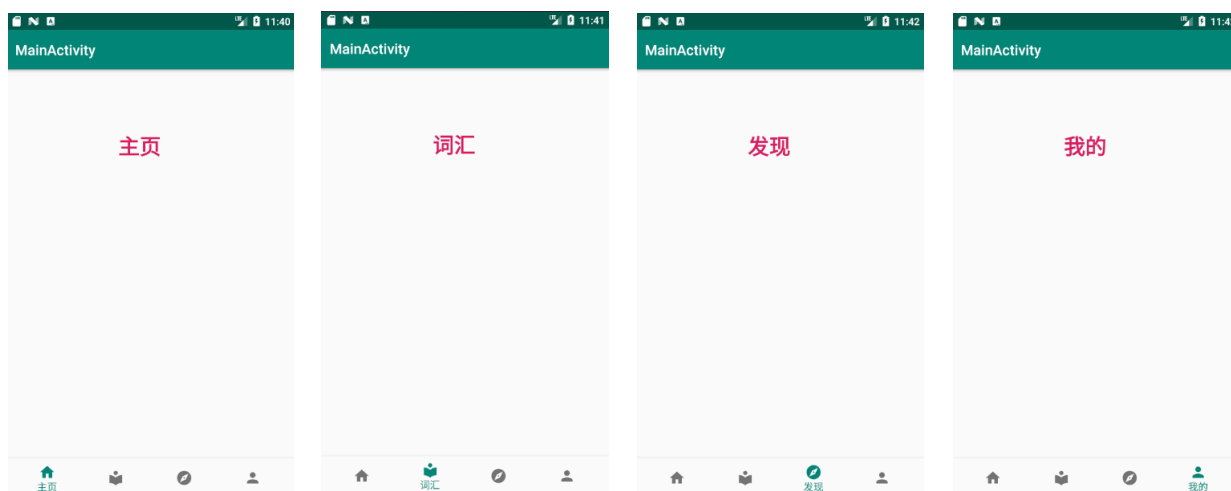


图 4-12 切换底部导航标签的运行效果

4.5 设备旋转与 Activity 生命周期

运行 NEC Vocab 应用，点击“我的”Tab，然后旋转设备，咦？怎么回到了主页？我们明明点击的是“我的”标签（Tab）页呀？！

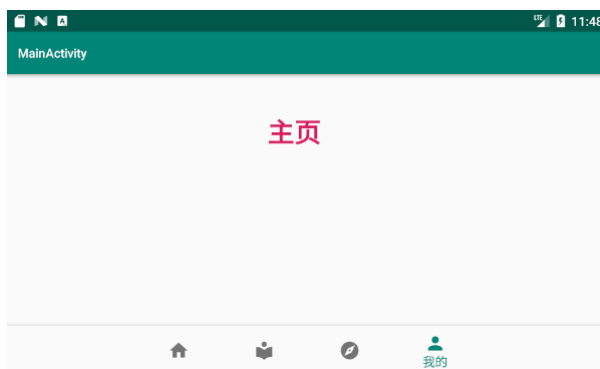


图 4-13 设备旋转后回到主页

设备旋转后，NEC Vocab 应用又回到了主页。

从第二章我们知道，旋转设备时操作系统依次调用了 Activity 的 `onPause()` → `onStop()` → `onDestroy()` → `onCreate(Bundle)` → `onStart()` → `onResume()` 方法。即：当设备旋转时，当前的 MainActivity 实例会被销毁，然后创建一个新的 MainActivity 实例。

在操作系统重新调用 `onCreate` 方法时，该方法调用了：

```
mTextMessage.setText(R.string.title_home);
```

即重新将 TextView 设置为默认的显示内容“主页”。

4.5.1 设备配置与备选资源

旋转设备会改变设备配置（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、键盘类型、底座模式以及语言，等等。通常，为匹配不同的设备配置，应用会提供不同的备选资源。

设备的屏幕密度是个固定的设备配置，无法在运行时发生改变。然而，有些特征，如屏幕方向，可以在应用运行时进行改变。

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。下面为设备配置变更新建备选资源，只要设备旋转至水平方位，Android 就会自动发现并使用它。

创建水平模式布局

由于在第二章我们已经为 NEC Vocab 应用创建了水平布局的目录 `layout-land`，现在只需要将布局文件 `activity_main.xml` 复制到该目录中即可。

接下来打开 `layout-land/activity_main.xml`，在 Design 标签页下修改该文件布局，为了与垂

直布局有所区分，我们将 `TextView` 放置在中央，并将字体设置为蓝色，将字号设置得更大，如图 4-14 所示（当然你可以设计一个更美观的布局）。

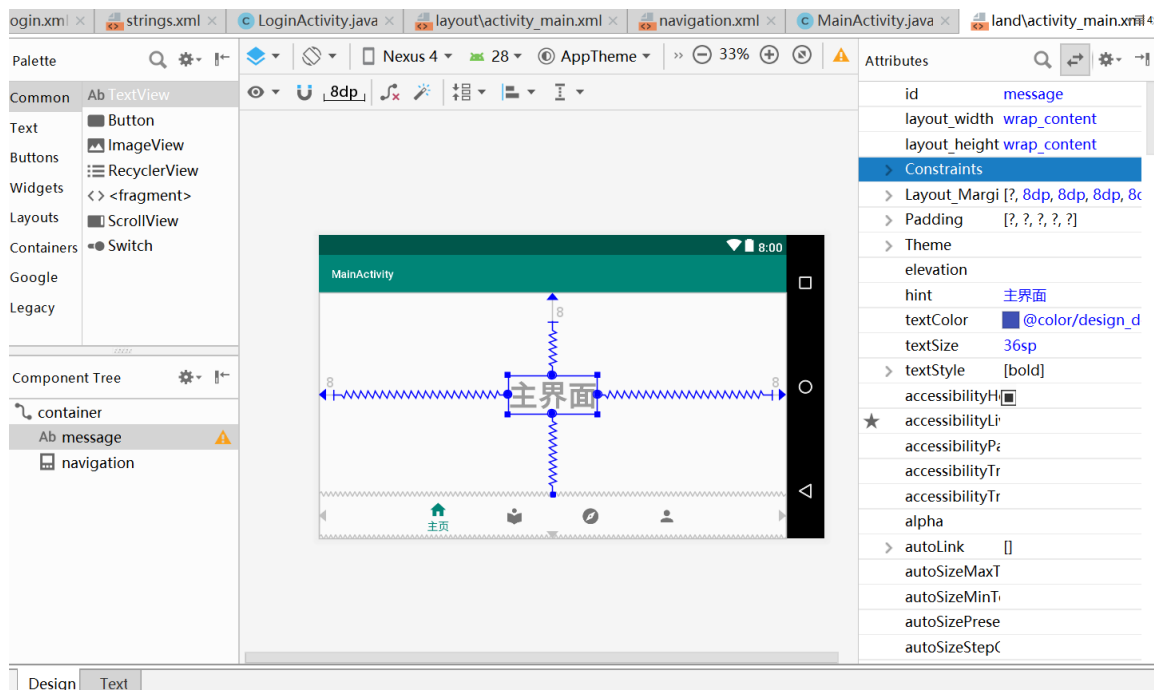


图 4-14 主界面水平布局

再次运行 NEC Vocab 应用。旋转设备至水平方位，查看新的布局界面，如图 4-15 所示。当然，这不仅是一个新的布局界面，也是一个新的 `MainActivity`。

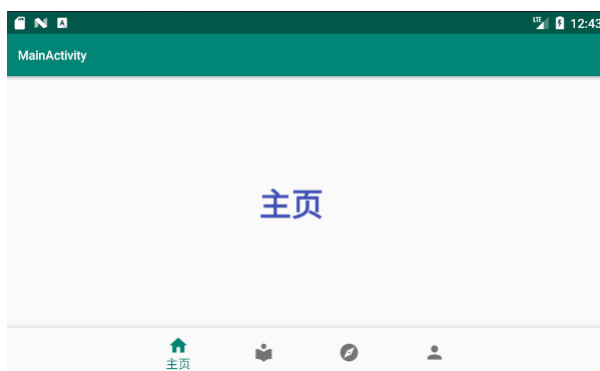


图 4-15 处于水平方位的 MainActivity

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

Android 可自动完成最佳匹配资源的调用，但前提是它必须通过新建一个 `activity` 来实现。`MainActivity` 要显示一个新布局，需再次调用 `setContentView(R.layout.activity_main)` 方法。而调用 `setContentView(R.layout.activity_main)` 方法又必须先调用 `MainActivity.onCreate(...)` 方法。因此，

设备一经旋转，Android 需要销毁当前的 MainActivity，然后新建一个 MainActivity 来完成 MainActivity.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，在应用运行中，只要设备配置发生了改变，Android 就会销毁当前 activity，然后再创建新的 activity。另外，虽然在应用运行中也会发生可用键盘或语言的变化，但设备屏幕方向的改变最为常见。

4.5.2 设备旋转前保存数据

适时使用备选资源虽然是 Android 提供的较完美的解决方案，但是，设备旋转导致的 activity 销毁与新建也会带来麻烦。比如，设备旋转后，NEC Vocab 应用回到“主页”的缺陷。

要修正这个缺陷，旋转后新创建的 MainActivity 需要知道底部导航栏哪个 Tab 被点击，以便在 onCreate(...)方法中设置 mTextMessage.setText(...)时将显示内容设置为该 Tab 所指示的那个值，即 item.getItemId()所指示的值。

因此，在设备运行中发生配置变更时，如设备旋转，需采用某种方式保存以前的数据。覆盖以下 Activity 方法就是一种实现方式：

```
protected void onSaveInstanceState(Bundle outState)
```

该方法通常在 onPause()、onStop()以及 onDestroy()方法之前由系统调用。

方法 onSaveInstanceState(...)的默认实现要求所有 activity 视图将自身状态数据保存在 Bundle 对象中。Bundle 是存储字符串键与限定类型值之间映射关系（键值对）的一种结构。如下列代码所示，Bundle 作为参数传入 onCreate(Bundle)方法：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
}
```

覆盖 onCreate(...)方法时，我们实际是在调用 activity 超类的 onCreate(...)方法，并传入收到的 bundle。在超类代码实现里，通过取出保存的视图状态数据，activity 的视图层级结构得以重新创建。

覆盖 onSaveInstanceState(Bundle)方法

可通过覆盖 onSaveInstanceState(...)方法，将一些数据保存在 bundle 中，然后在 onCreate(...)方法中取回这些数据。处理设备旋转问题时，将采用这种方式保存 item.getItemId()所指示的值。

首先，打开 MainActivity.java 文件，新增一个常量 `index` 和一个变量 `mCurrentIndex` 作为将要存储在 `bundle` 中的键值对的键和值，并让 `mCurrentIndex`“记住”`item.getItemId()`所指示的值。如代码清单 4-7 所示。

代码清单 4-7 新增键值对的键和值（MainActivity.java）

```
public class MainActivity extends AppCompatActivity {
    private static final String KEY_INDEX = "index";
    private int mCurrentIndex = 0;
    private TextView mTextMessage;

    private BottomNavigationView.OnNavigationItemSelectedListener mOnNavigationItemSelectedListener
        = new BottomNavigationView.OnNavigationItemSelectedListener() {
        @Override
        public boolean onNavigationItemSelected(@NonNull MenuItem item) {
            mCurrentIndex = item.getItemId();
            switch (item.getItemId()) {
                ....
            }
        }
    };
}
```

然后，覆盖 `onSaveInstanceState(...)`方法，以刚才新增的常量值作为键，将 `mCurrentIndex` 变量值保存到 `bundle` 中，如代码清单 4-8 所示。

代码清单 4-8 覆盖 `onSaveInstanceState(...)`方法（MainActivity.java）

```
public class MainActivity extends AppCompatActivity {
    private static final String KEY_INDEX = "index";
    private int mCurrentIndex = 0;
    ....
    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
    }
}
```

最后，在 `onCreate(...)`方法中确认是否成功获取该数值。如获取成功，则将它赋值给变量 `mCurrentIndex`，再根据 `mCurrentIndex` 的值来确定显示内容，如代码清单 4-9 所示（这里给出全部参考代码，以便读者核对）。

代码清单 4-9 在 `onCreate(...)`方法中检查存储的 `bundle` 信息（MainActivity.java）

```
public class MainActivity extends AppCompatActivity {
    private static final String KEY_INDEX = "index";
    private int mCurrentIndex=0;
    private TextView mTextMessage;

    private BottomNavigationView.OnNavigationItemSelectedListener mOnNavigationItemSelectedListener
        = new BottomNavigationView.OnNavigationItemSelectedListener() {
        @Override
        public boolean onNavigationItemSelected(@NonNull MenuItem item) {
            ....
        }
    };
}
```

```

public boolean onNavigationItemSelected(@NonNull MenuItem item) {
    mCurrentIndex = item.getItemId();
    switch (mCurrentIndex) {
        case R.id.navigation_home:
            mTextMessage.setText(R.string.title_home);
            return true;
        case R.id.navigation_voclib:
            mTextMessage.setText(R.string.title_voclib);
            return true;
        case R.id.navigation_discover:
            mTextMessage.setText(R.string.title_discover);
            return true;
        case R.id.navigation_mine:
            mTextMessage.setText(R.string.title_mine);
            return true;
    }
    return false;
}
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mTextMessage = (TextView) findViewById(R.id.message);
    mTextMessage.setText(R.string.title_home);
    mCurrentIndex = R.id.navigation_home;
    if (savedInstanceState != null){
        mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
    }

    switch (mCurrentIndex) {
        case R.id.navigation_home:
            mTextMessage.setText(R.string.title_home);
            break;
        case R.id.navigation_voclib:
            mTextMessage.setText(R.string.title_voclib);
            break;
        case R.id.navigation_discover:
            mTextMessage.setText(R.string.title_discover);
            break;
        case R.id.navigation_mine:
            mTextMessage.setText(R.string.title_mine);
            break;
    }

    BottomNavigationView navigation = (BottomNavigationView) findViewById(R.id.navigation);
    navigation.setOnNavigationItemSelectedListener(mOnNavigationItemSelectedListener);
}

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
}

```

运行 NEC Vocab 应用，单击底部导航栏的各个 Tab。现在，无论设备自动或手动旋转多少次，新创建的 MainActivity 都会记住当前 Tab 标签正在显示的内容。

注意，在 Bundle 中存储和恢复的数据类型只能是基本数据类型（primitive type）以及可以实现 Serializable 或 Parcelable 接口的对象。在 Bundle 中保存定制类对象不是个好主意，因为你取回的对象可能已经过时了。比较好的做法是，通过其他方式保存定制类对象，而在 Bundle 中保存对象对应的基本数据类型的标示符。

测试 onSaveInstanceState(...)实现方法是个好习惯，尤其在需要存储和恢复对象时。设备旋转很容易测试，但测试低内存状态就困难多了。本章末尾会深入学习这部分内容，继而学习如何模拟 Android 为回收内存而销毁 activity 的场景。

4.5.3 再探 Activity 生命周期

覆盖 onSaveInstanceState(...)方法并不仅仅用于处理设备旋转相关的问题。用户离开当前 activity 管理的用户界面，或 Android 需要回收内存时，activity 也会被销毁。

基于用户体验考虑，Android 从不会为了回收内存，而去销毁正在运行的 activity。activity 只有在暂停或停止状态下才可能会被销毁。此时，会调用 onSaveInstanceState(...)方法。

调用 onSaveInstanceState(...)方法时，用户数据随即被保存在 Bundle 对象中。然后操作系统将 Bundle 对象放入 activity 记录中。为便于理解 activity 记录，我们增加一个暂存状态（stashed state）到 activity 生命周期，如图 4-16 所示。activity 暂存后，Activity 对象不再存在，但操作系统会将 activity 记录对象保存起来。这样，在需要恢复 activity 时，操作系统可以使用暂存的 activity 记录重新激活 activity。注意，activity 进入暂存状态并不一定需要调用 onDestroy()方法。不过，onPause()和 onSaveInstanceState(...) 通常是我们需要调用的两个方法。常见的做法是，覆盖 onSaveInstanceState(...)方法，将数据暂存到 Bundle 对象中，覆盖 onPause()方法处理其他需要处理的事情。

有时，Android 不仅会销毁 activity，还会彻底停止当前应用的进程。不过，只有在用户离开当前应用时才会发生这种情况。即使这种情况真的发生了，暂存的 activity 记录依然被系统保留着，以便于用户返回应用时 activity 的快速恢复。

那么暂存的 activity 记录到底可以保留多久？前面说过，用户按了后退键后，系统会彻底销毁当前的 activity。此时，暂存的 activity 记录同时被清除。此外，系统重启或长时间不使用 activity 时，暂存的 activity 记录通常也会被清除。

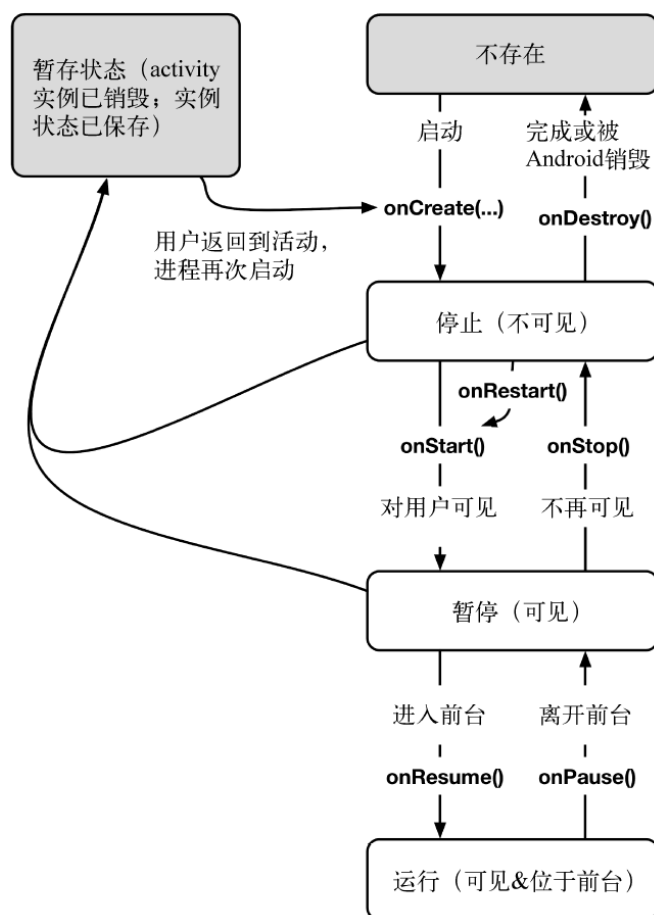


图 4-16 完整的 activity 生命周期

4.6 Android 应用的调试

本节将学习如何处理应用 bug，同时也将学习如何使用 LogCat 和 Debug 调试代码。

为练习调试，我们先刻意搞点破坏。打开 MainActivity.java 文件，在 onCreate(Bundle)方法中，注释掉获取 TextView 组件的那行代码，如代码清单 4-10 所示。

代码清单 4-10 注释掉一行关键代码 (**MainActivity.java**)

```

.....
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //mTextMessage = (TextView) findViewById(R.id.message);
    .....
}
.....

```

运行 NEC Vocab 应用，看看会发生什么。

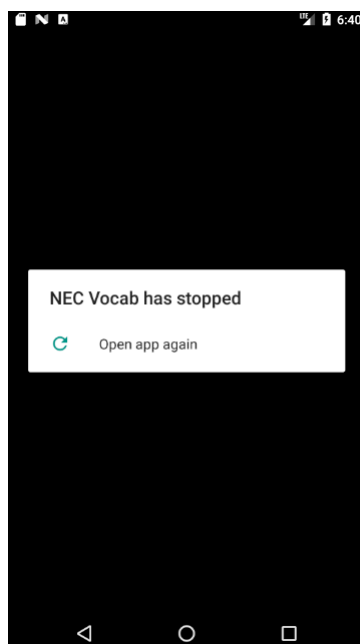


图 4-17 NEC Vocab 应用崩溃了

图 4-17 是应用崩溃后的消息提示画面。不同 Android 版本的消息提示可能略有不同，但本质上它们都是同一个意思。当然，这里我们知道应用为何崩溃。但假如不知道的话，从另一个角度来看或许有助于问题的排查。

4.6.1 异常与栈跟踪

为方便查看异常或错误信息，展开 Android DDMS 工具窗口。上下滑动 LogCat 窗口滚动条，应该会看到如图 4-18 所示的整片红色的异常或错误信息。这就是标准的 Android 运行时的异常信息报告。如果看不到，可试着选择 LogCat 的 No Filters 过滤项。另外，还可以调整 Log Level 为 Error，让系统只输出严重问题日志。

```
2018-11-04 14:47:47.319 21494-21494/com.studio.aime.necvocab E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.studio.aime.necvocab, PID: 21494
java.lang.NullPointerException: Attempt to invoke virtual method 'void android.widget.TextView.setText(int)' on a null object reference
    at com.studio.aime.necvocab.MainActivity$1.onNavigationItemSelectedListener(MainActivity.java:23)
    at android.support.design.widget.BottomNavigationView$1.onMenuItemSelected(BottomNavigationView.java:204)
    at android.support.v7.view.menu.MenuBuilder.dispatchMenuItemSelected(MenuBuilder.java:840)
    at android.support.v7.view.menu.MenuItemImpl.invoke(MenuItemImpl.java:158)
    at android.support.v7.view.menu.MenuBuilder.performItemAction(MenuBuilder.java:991)
    at android.support.design.internal.BottomNavigationView$1.onClick(BottomNavigationView.java:115)
    at android.view.View.performClick(View.java:5610)
    at android.view.View$PerformClick.run(View.java:22265)
    at android.os.Handler.handleCallback(Handler.java:731)
    at android.os.Handler.dispatchMessage(Handler.java:95)
    at android.os.Looper.loop(Looper.java:134)
    at android.app.ActivityThread.main(ActivityThread.java:6077) <1 internal call>
```

图 4-18 LogCat 中的异常与栈追踪

该异常报告首先告诉了我们最高层级的异常及其栈追踪，然后是导致该异常的异常及其栈追踪。如此不断追溯，直到找到一个没有原因的异常。

在我们编写的大部分代码中，最后一个没有原因的异常往往是要关注的目标。这里，没有原因的异常是 `java.lang.NullPointerException`。紧接着该异常语句的一行就是其栈追踪信息的第一行。从该行可以看出发生异常的类和方法以及它所在的源文件及代码行号。单击蓝色链接，Android Studio 会自动跳转到源代码的对应行上。

Android Studio 定位的这行代码是 `mTextMessage` 变量在 `onNavigationItemSelected(...)` 方法中的首次使用。名为 `NullPointerException` 的异常暗示了问题所在，即变量没有初始化。

为修正该问题，取消对变量 `mTextMessage = (TextView) findViewById(R.id.message)` 初始化语句的注释。

遇到运行异常时，记得在 LogCat 中寻找最后一个异常及其栈追踪的第一行（该行对应着源代码）。这里是问题发生的地方，也是寻找问题答案的最佳起点。

如果发生应用崩溃的设备没有连接到电脑上，日志信息也不会全部丢失。设备会将最近的日志保存到 log 文件中。日志文件的内容长度及保留的时间取决于具体的设备，不过，获取十分钟之内产生的日志信息通常是有保证的。只要将设备连上电脑，在 Devices 视图里选择所用设备，LogCat 将自动打开并显示日志文件保存的内容。

诊断应用异常

应用出错不一定总会导致应用崩溃。某些时候，应用只是出现了运行异常。例如，在注册代码里，由于某种原因，我们将“再次输入密码”的字符串不小心设置为“手机号码”的字符串：

```
String passwordCheck = mMobile.getText().toString();
```

此时，即使在“再次输入密码”栏输入了正确的密码，单击底部的“确定”按钮时，应用都将提示：“两次密码输入不一样”。这就是一个非崩溃型的应用运行异常。

在 `RegisterActivity.java` 中，修改 `checkInputAndPermission()` 的代码，如代码清单 4-11 所示。

代码清单 4-11 注释掉一行关键代码（`RegisterActivity.java`）

```
.....
private boolean checkInputAndPermission() {
    //检查，重置错误
    mUserName.setError(null);
    mMobile.setError(null);
    mPassword.setError(null);
    mPasswordCheck.setError(null);
}
```

```
// 在注册完成后点击确定时尝试存储值。
```

```
String username = mName.getText().toString();
String mobile = mMobile.getText().toString();
String password = mPassword.getText().toString();
String passwordCheck = mMobile.getText().toString();//mPasswordCheck.getText().toString();
```

```
.....
```

运行 NEC Vocab 应用，进入注册界面，录入相关信息，点击“确定”按钮。可以看到，应用总是出现“**两次密码输入不一样**”提示。

这个问题要比上个更为棘手。它没有抛出异常，所以修正这个问题不像前面跟踪追溯并消除异常那么简单。有了解决上个问题的经验，这里可以推测出导致该问题的两种可能因素：

- passwordCheck 变量值没有设置成功；
- password.equals(passwordCheck)==false 总是为 true。

如确实不知道问题产生的原因，则需要设法跟踪并找出问题所在。在接下来的几小节里，我们将学习到两种跟踪问题的方法：

- 记录栈跟踪的诊断性日志；
- 利用调试器设置断点调试。

4.6.2 记录栈跟踪日志

在 RegisterActivity 中，添加一条 TAG 的设置语句：String TAG = "RegisterActivity"，同时为 checkInputAndPermission() 方法添加日志输出语句，如代码清单 4-12 所示。

代码清单 4-12 方便实用的调试方式（RegisterActivity.java）

```
public class RegisterActivity extends AppCompatActivity {
    private static final String TAG = "RegisterActivity";
    .....
    private boolean checkInputAndPermission() {
        .....
        String passwordCheck = mMobile.getText().toString();//mPasswordCheck.getText().toString();
        .....
        //判断两次密码输入是否相同，如果不同，提示密码不一样
        Log.d(TAG, "验证密码信息显示 #" +
            mPasswordCheck.getText().toString(), new Exception());
        if(password.equals(passwordCheck)==false){
            Toast.makeText(this, getString(R.string.pwd_not_the_same), Toast.LENGTH_SHORT).show();
            return false;
        }
        .....
    }
}
```


如同前面 `AndroidRuntime` 的异常, `Log.d(String, String, Throwable)`方法记录并输出整个栈跟踪信息。借助栈跟踪日志, 可以看出 `checkInputAndPermission()`方法在哪些地方被调用了。

作为参数传入 `Log.d(...)`方法的异常不一定是我们捕获的已抛出异常。可以创建一个新的 `Exception()`方法, 把它作为不抛出的异常对象传入该方法。借此, 我们得到异常发生位置的记录报告。

运行 `NEC Vocab` 应用, 点击“我的”按钮, 然后在 `LogCat` 中查看日志输出, 输出结果如图 4-19 所示。

```
2018-11-04 16:29:58.445 23826-23826/com.studio.aime.necvocab D/RegisterActivity: 验证密码信息显示 #123
java.lang.Exception
    at com.studio.aime.necvocab.RegisterActivity.checkInputAndPermission(RegisterActivity.java:97)
    at com.studio.aime.necvocab.RegisterActivity.access$000(RegisterActivity.java:18)
    at com.studio.aime.necvocab.RegisterActivity$1.onClick(RegisterActivity.java:39)
    at android.view.View.performClick(View.java:5610)
    at android.view.View$PerformClick.run(View.java:22265)
    at android.os.Handler.handleCallback(Handler.java:731)
    at android.os.Handler.dispatchMessage(Handler.java:95)
    at android.os.Looper.loop(Looper.java:134)
```

图 4-19 日志输出结果

栈跟踪日志的第一行（蓝色）即调用异常记录方法的地方。紧跟着的两行表明 `checkInputAndPermission()`方法是在 `onClick(...)`实现方法里被调用的。双击该行即可跳转至该代码行。暂时保留该代码问题, 下一节我们会使用设置断点调试的方法重新查找该问题。

记录栈跟踪日志虽然是个强大的工具, 但也存在缺陷。比如, 大量的日志输出很容易导致 `LogCat` 窗口信息混乱难读。此外, 通过阅读详细直白的栈跟踪日志并分析代码意图, 竞争对手可以轻易剽窃我们的创意。

另一方面, 既然有时可以从栈跟踪日志看出代码的实际使用意图, 在网站或者论坛上寻求帮助时, 附上一段栈跟踪日志往往有助于更快地解决问题。如果需要, 我们可以直接从 `LogCat` 中复制并粘贴日志内容。

在继续学习之前, 先删除日志记录代码。

4.6.3 设置断点

要使用 `Android Studio` 自带的代码调试器调试上一节中我们遇到的问题, 首先要在 `checkInputAndPermission()`方法中设置断点, 以确认该方法是否被调用。断点会在断点设置行的前一行代码处停止运行, 然后我们可以逐行检查代码, 看看接下来到底发生了什么。

在 `RegisterActivity.java` 文件的 `checkInputAndPermission()`方法中, 双击第一行代码左边的灰

色栏区域。

可以看到，灰色栏上出现了一个红色圆圈。这就是我们设置的一处断点，如图 4-20 所示。

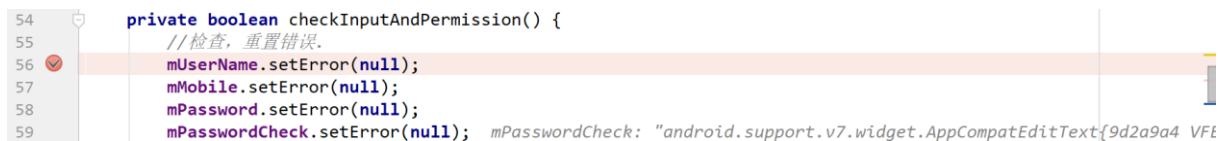




图 4-20 已设置的一处断点

启用代码调试器并触发已设置的断点，我们需要调试运行而不是直接运行应用。要调试运行应用，单击  按钮旁边的  按钮，或选择 Run → Debug 'app' 菜单项。设备会报告说正在等待调试器加载，然后继续运行。

进入注册界面，录入相关信息后，点击“确定”按钮，应用启动并加载调试器运行后，就会暂停，接着调用 checkInputAndPermission() 方法，然后触发断点。

如图 4-21 所示，RegisterActivity.java 代码已经在代码编辑区打开了，断点设置所在行的代码也被加亮显示了。应用在断点处停止了运行。

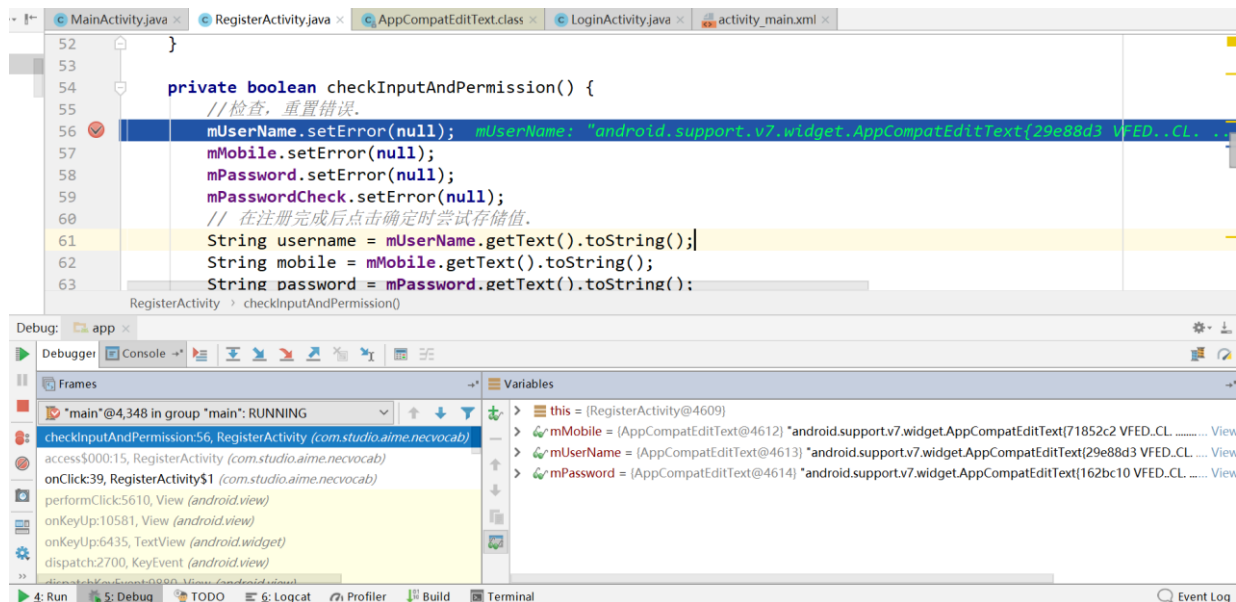


图 4-21 代码在断点处停止执行

这时，由 Frames 和 Variables 视图组成的 Debug 工具窗口出现在了 Android Studio 的底部，如图 4-22 所示。

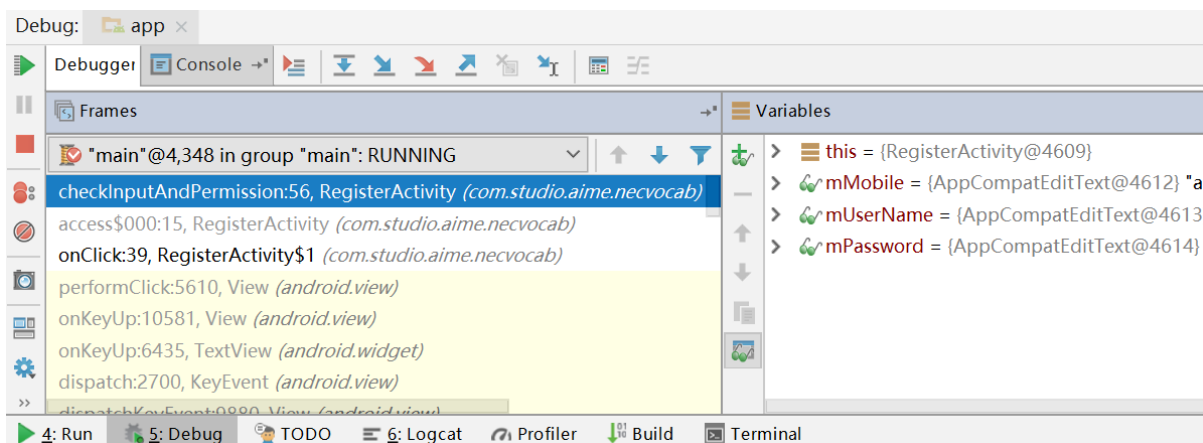




图 4-22 代码调试视图

使用调试视图顶部的  箭头按钮可单步执行应用代码。从栈列表可以看出 `checkInputAndPermission()` 方法已经在 `onClick(...)` 方法中被调用了。不过，我们需要关心的是检查“确定”按钮被点击后的行为。因此单击右边的  (Resume Program) 按钮让程序继续运行。然后，再次点击“确定”按钮观察断点是否被激活（应该激活）。

既然程序执行停在了断点处，就可以趁机了解其他视图。变量视图 (Variables) 可以让我们观察到程序中各对象的值。应该可以看到在 `RegisterActivity` 中创建的变量，另外还有一个特别的 `this` 变量值 (`RegisterActivity` 本身)。

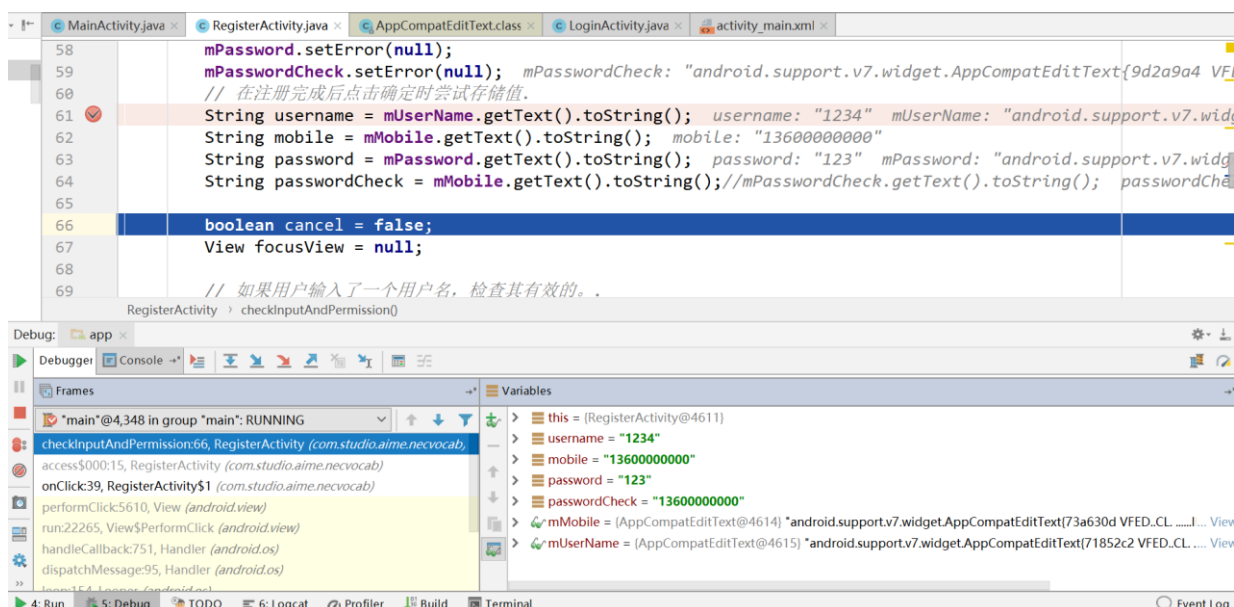


图 4-23 单步执行，找到问题的所在

展开 `this` 变量后可看到很多变量。它们是 `RegisterActivity` 类的 `Activity` 超类、`Activity` 超类

的超类（一直追溯到继承树的顶端）的全部变量。

我们只需关心变量 `passwordCheck` 的值。继续单步执行，当执行到如图 4-23 所示的代码时，我们发现 `passwordCheck` 的值为“13600000000”。与 `password` 的值“123”，以及我们在“再次输入密码”栏中输入的“123”不同。进一步考察发现，这是 `mobile` 的输入值。

真是相当方便的调试，问题解决了。