

Android 软件开发

Android 开发环境及 Android 工程简介

秦兴国

xgqin@guet.edu.cn

计算机与信息安全学院
桂林电子科技大学

2017 年 10 月 22 日

1 Android 开发环境简介

2 第一个 App — *Hello World*

- 创建工程
- 配置虚拟设备
- 运行工程

3 通过 *Hello world* 了解 Android 工程结构

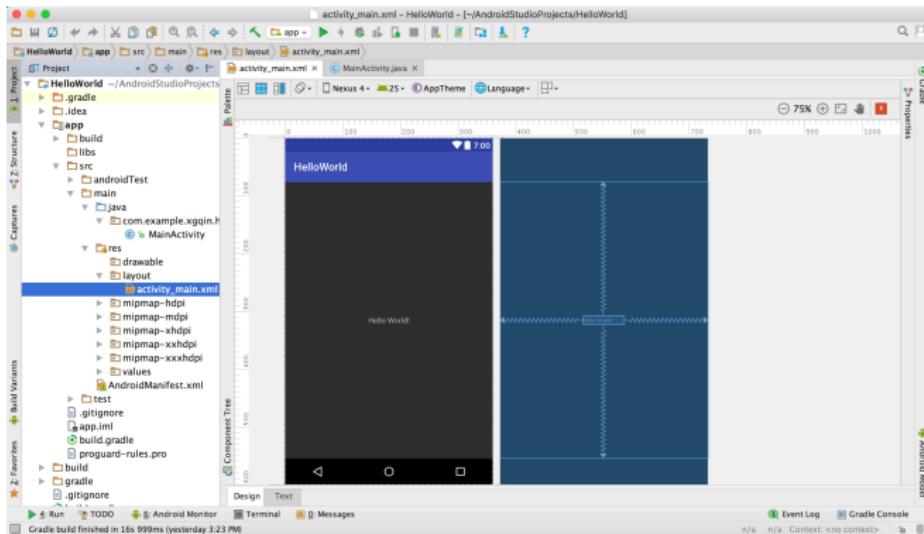
- Android 工程文件结构介绍
- How it works? — *Hello world*
- 改进一下 — Hello World

4 App Manifest(系统清单) 与 `AndroidManifest.xml`

5 Intent 与 Intent Filters

The Official IDE for Android — Android Studio

Android Studio¹ is the official Integrated Development Environment (IDE) for Android app development, based on **IntelliJ IDEA²**



¹ <https://developer.android.com/studio/index.html>

² <https://www.jetbrains.com/idea/>

Features of Android Studio I

On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- Instant Run³
- Intelligent code editor
- Fast and feature-rich emulator
- Robust and flexible build system
- Testing tools & frameworks
- Layout Editor

³ <http://www.jianshu.com/p/2e23ba9ff14b>

Features of Android Studio II



Instant Run

Android Studio's Instant Run feature pushes code and resource changes to your running app.

It intelligently understands the changes and often delivers them without restarting your app or rebuilding your APK, so you can see the effects immediately.



Instant Run

Android Studio's Instant Run feature pushes code and resource changes to your running app.

It intelligently understands the changes and often delivers them without restarting your app or rebuilding your APK, so you can see the effects immediately.



更快及包含丰富特性的模拟器

Android 模拟器在安装及启动 App 时比物理机更快，从而更好的支持在不同设备配置下 (手机，平板，穿戴设备，以及 Android TV) 的 App 原型开发及测试；

Android 模拟器对众多硬件特性进行模拟，例如：GPS 位置传感器，网络延迟，运动传感器，多点触控输入等；



Robust and flexible build system

Android Studio offers build automation, dependency management, and customizable build configurations. You can configure your project to include local and hosted libraries, and define build variants that include different code and resources, and apply different code shrinking and app signing configurations.

Features of Android Studio III



Robust and flexible build system

Android Studio offers build automation, dependency management, and customizable build configurations.

You can configure your project to include local and hosted libraries, and define build variants that include different code and resources, and apply different code shrinking and app signing configurations.

Intelligent code editor

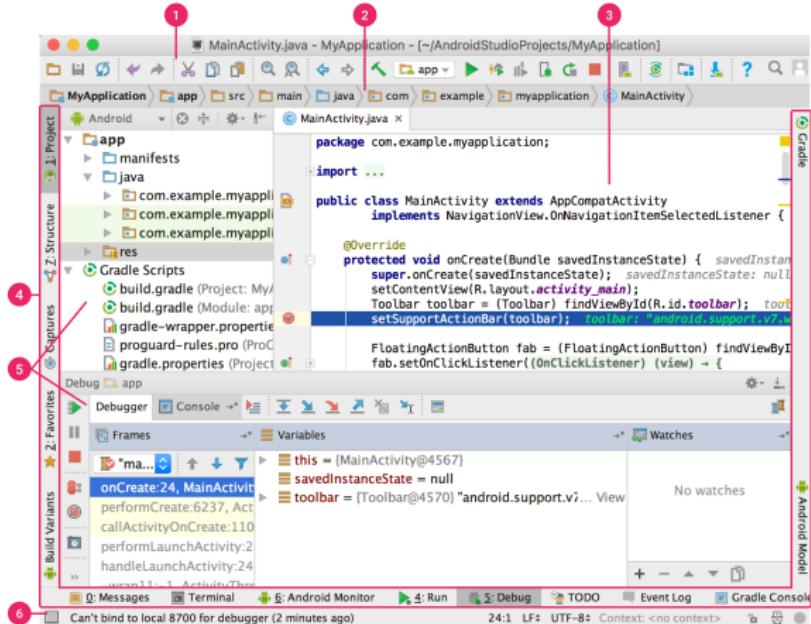


The code editor helps you write better code, work faster, and be more productive by offering advanced code completion, refactoring, and code analysis.

As you type, Android Studio provides suggestions in a dropdown list. Simply press Tab to insert the code.

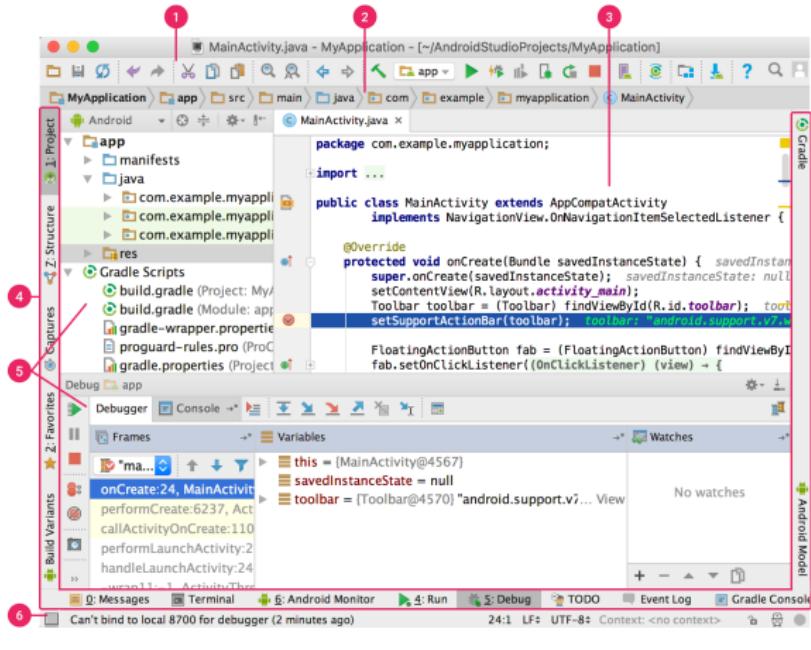
Android Studio 启动及界面功能

Android Studio 主界面如下所示，主要可分为如下几个部分：



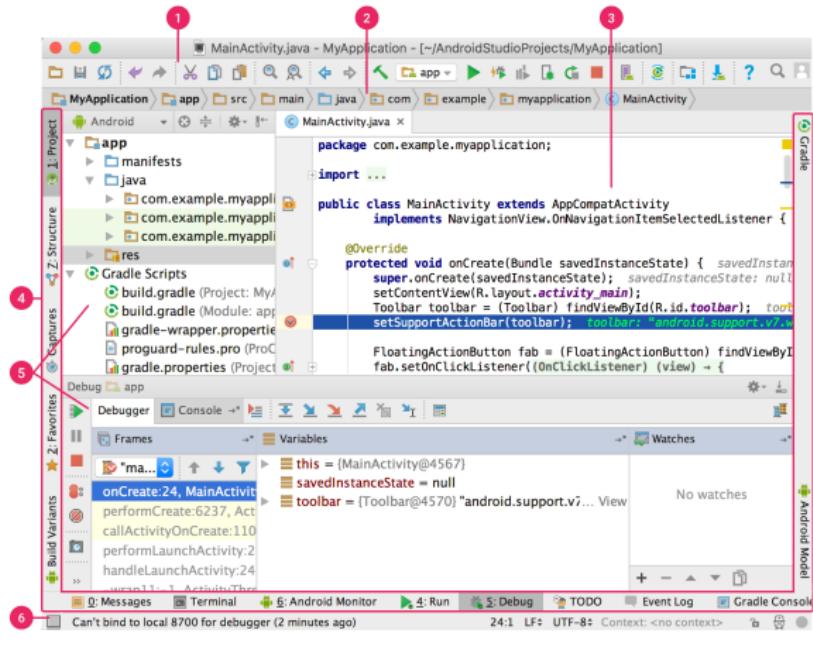
Android Studio 启动及界面功能

① 工具栏 (toolbar) 通常包括常用的操作按钮，例如：运行、调试 App，管理 Android 模拟器设备等；



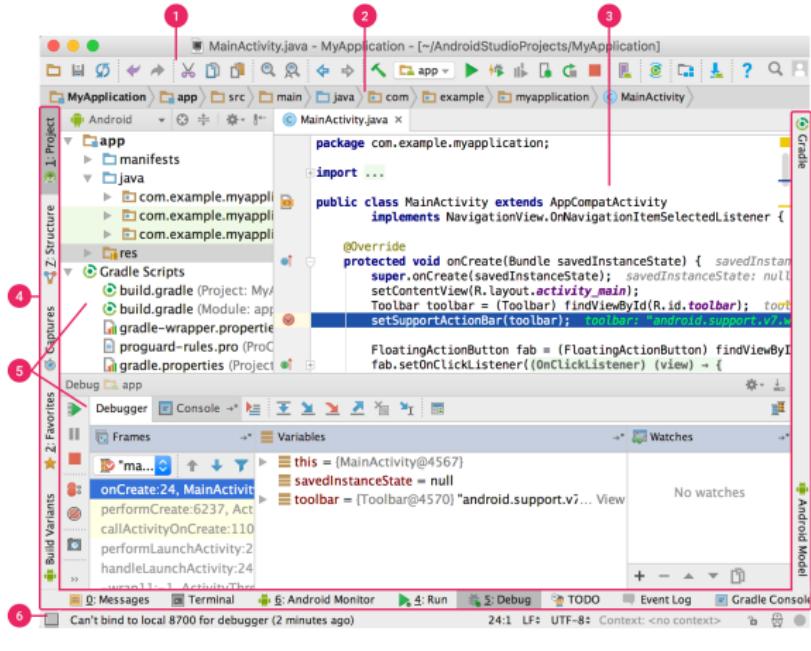
Android Studio 启动及界面功能

② 导航栏 (navigation bar) 可以帮助开发者对当前工程进行浏览及文件打开等操作。相对于工程窗口而言，其提供了更为简洁直观方式显示工程结构；



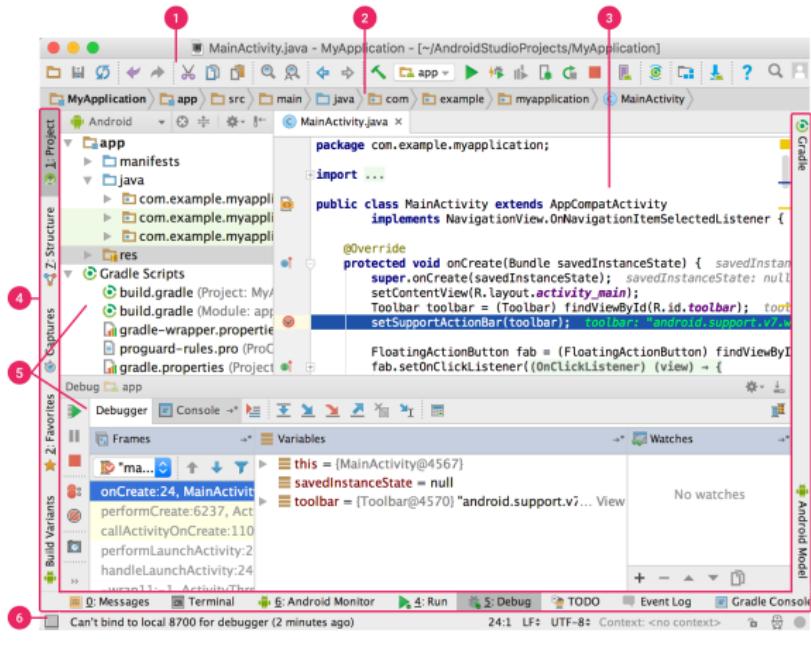
Android Studio 启动及界面功能

③ 编辑窗口 (editor window) 主要用于编辑代码。编辑窗口根据当前打开文件类型，启动不同编辑器对文件进行编辑操作。例如，代码编辑器对源代码进行编辑，布局编辑器对布局文件进行编辑等；



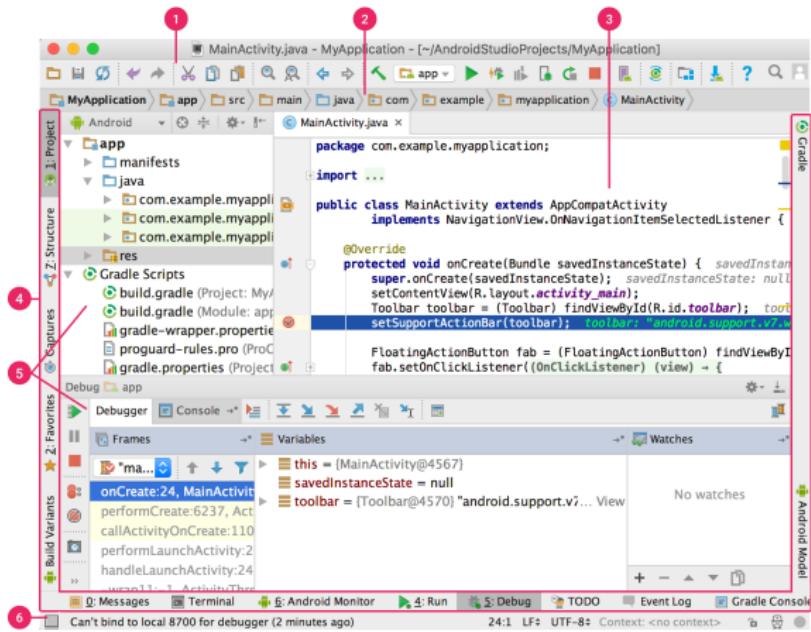
Android Studio 启动及界面功能

④ 工具窗口栏 (tool window bar) 停靠在 IDE 主窗口四周，为开发者提供独立的各类工具窗口，开发者可以对这类窗口进行展开或收缩操作；



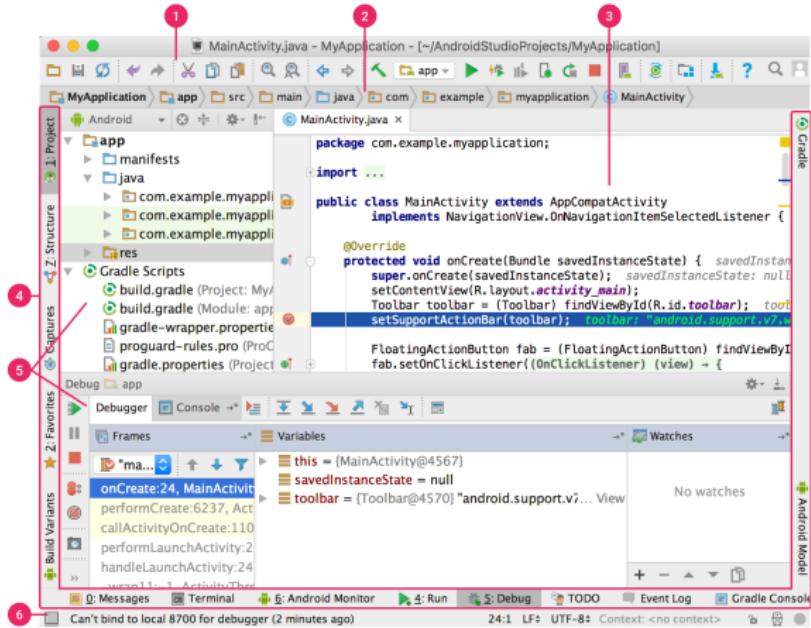
Android Studio 启动及界面功能

⑤ 工具窗口 (tool windows) 为开发者便捷地提供诸如工程管理, 查找、源码版本控制等特定功能;



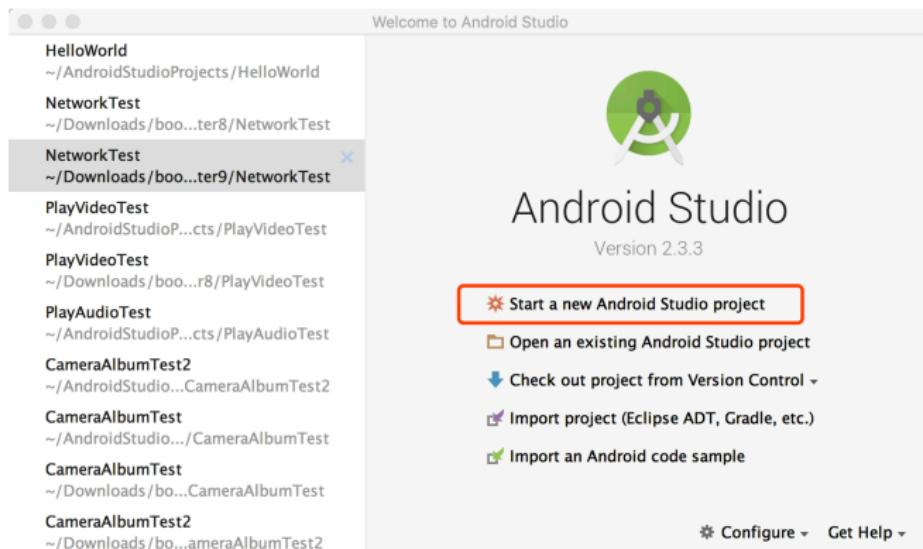
Android Studio 启动及界面功能

⑥ 状态栏 (status bar) 主要显示当前工程或 IDE 状态, 以及警告等相关信息;



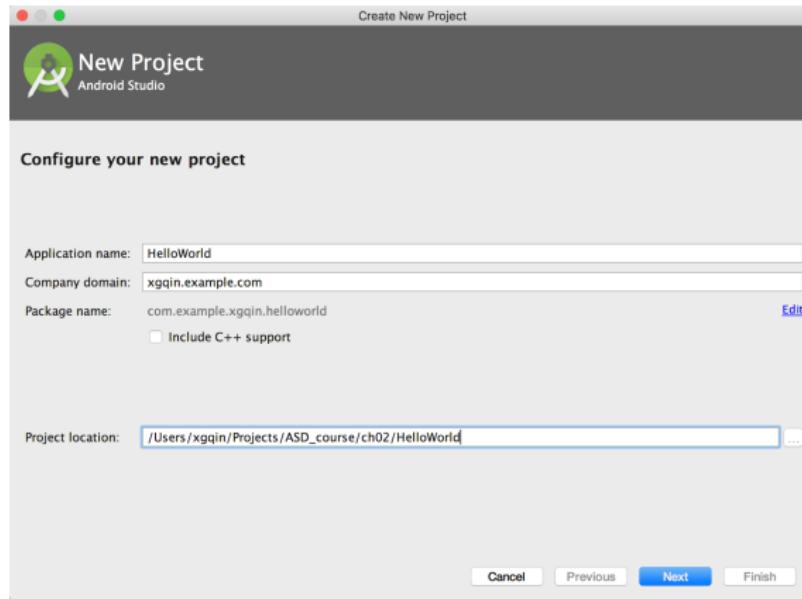
启动 Android Studio

启动 Android Studio，在 Android 启动界面中，选择“**start a new Android Studio project**”；



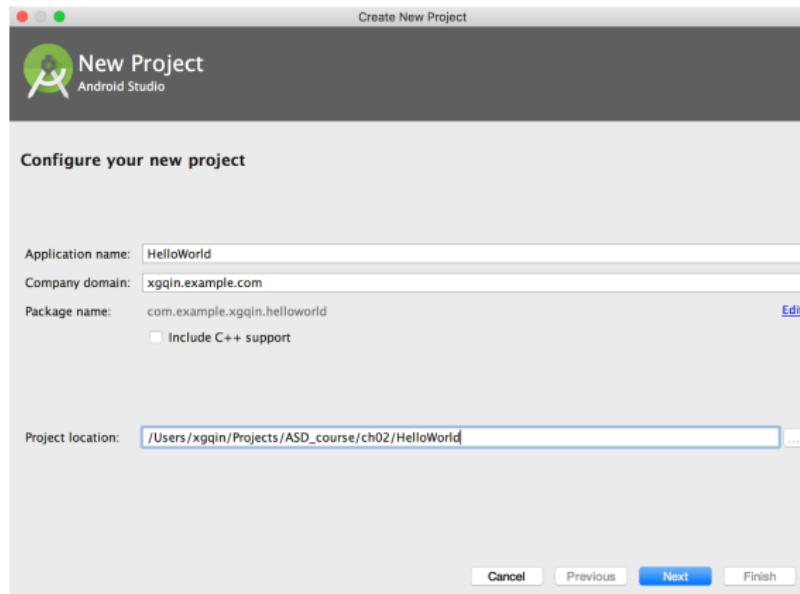
创建新 Android Studio 工程

Application name: 工程名, 应用程序名;



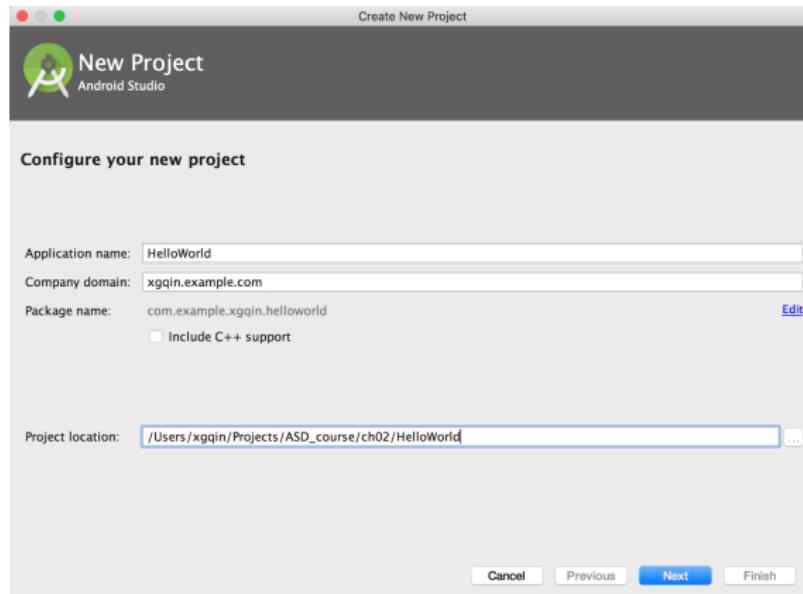
创建新 Android Studio 工程

Company domain: 公司域名，用于后续源码中各包 (package) 的命名，以及应用包名 (App package name) 中；



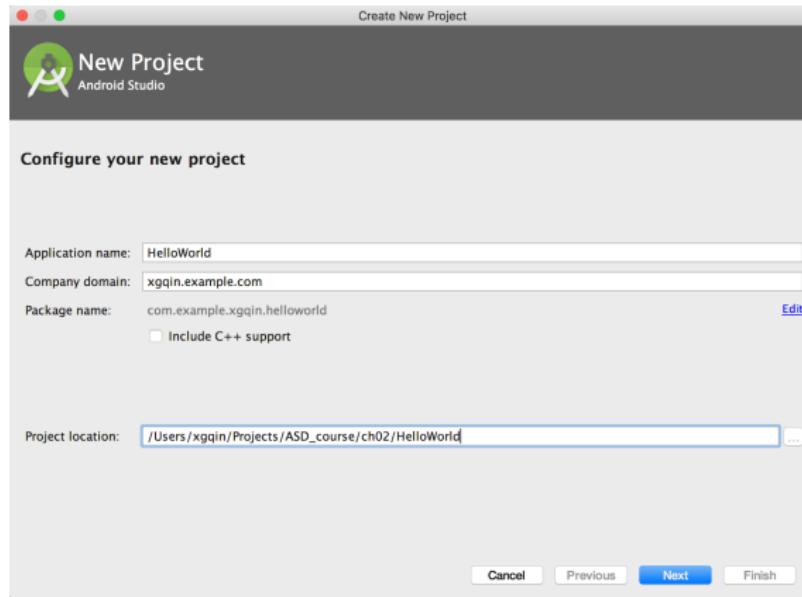
创建新 Android Studio 工程

Package name: 应用包名，通常以公司域名为逆序 + 应用程序名，**作为应用 APK 包的唯一标示**，应该与其他 APK 不同；



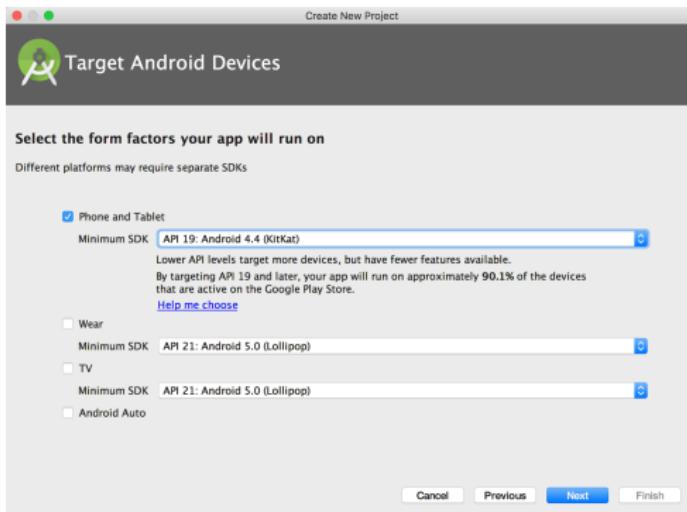
创建新 Android Studio 工程

Project location: 工程存储路径;



应用程序运行的目标 Android 设备

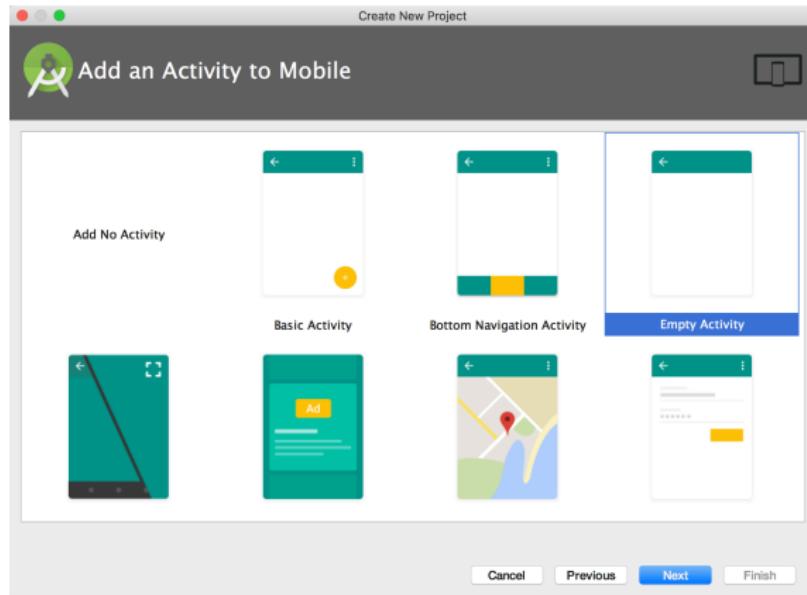
Minimum SDK: 开发的 App 能支持的最低 Android SDK 版本；



▶ Help me choose

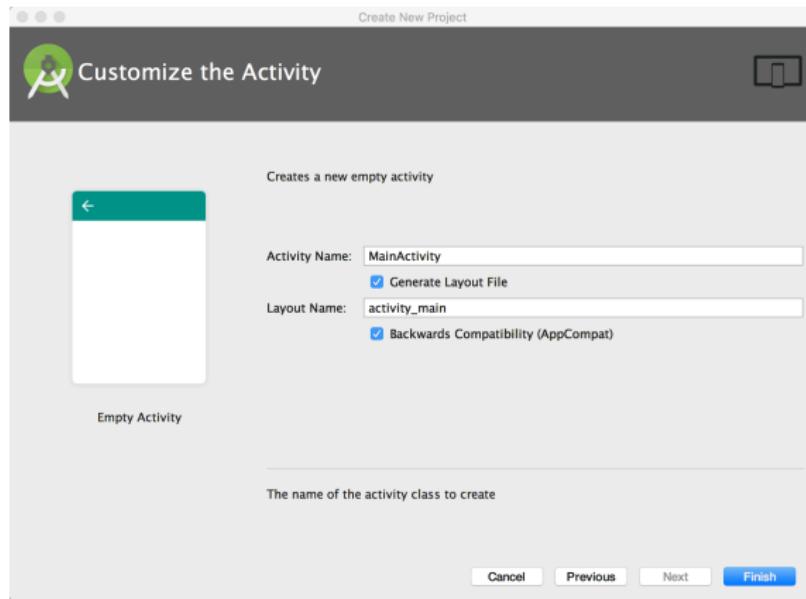
为应用添加 Activity

Android Studio 创建工程向导，提供多种模板 Activity 供用户选择；



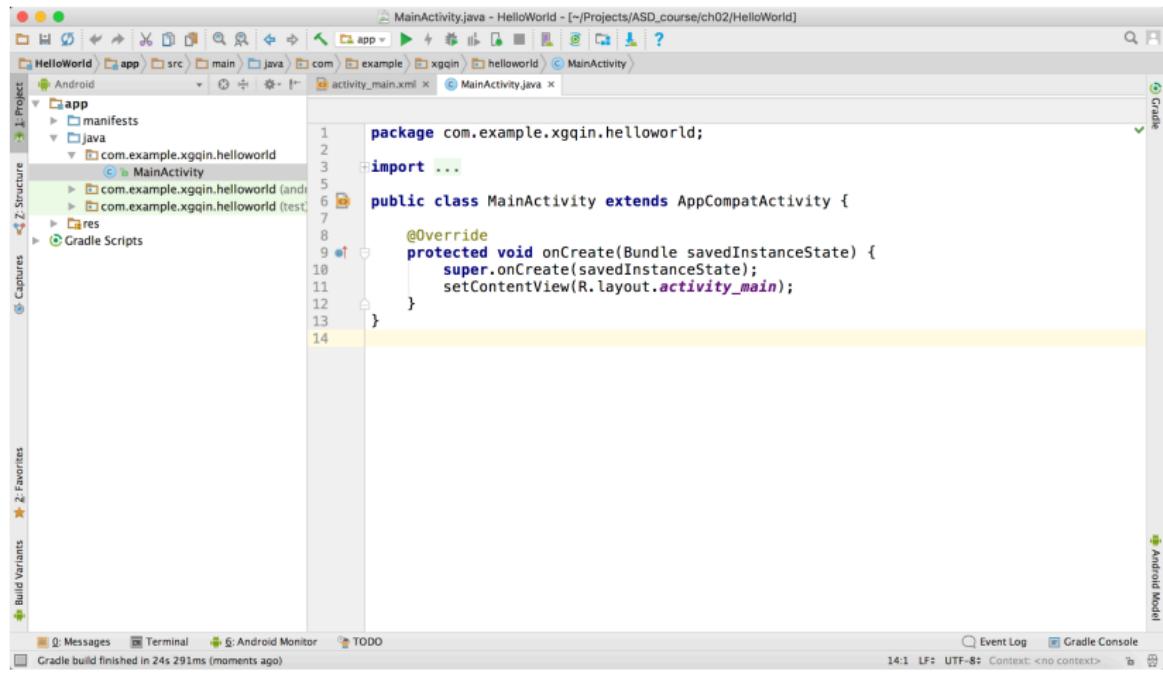
配置 Activity

设置要添加进工程的 Activity 名及其对应的布局文件名；



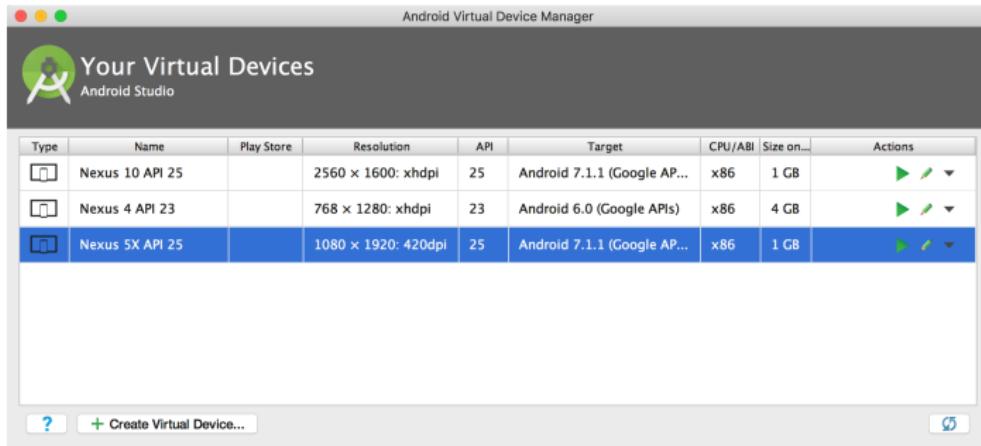
完成工程创建

完成工程创建后，Android Studio 进入主界面，并在相关窗口显示工程及源码信息；



启动 AVD Manager

在 Android Studio 主界面工具栏中，点击  图标，启动“AVD Manager”对虚拟设备进行管理；

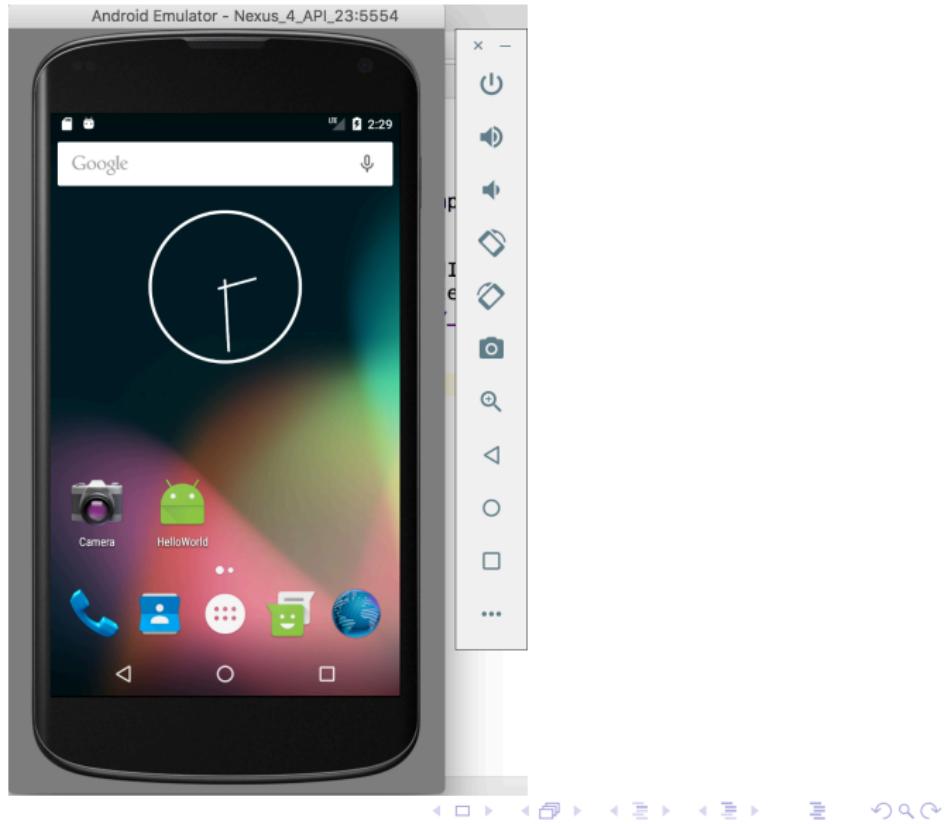


在 AVD Manager 中选择需要运行的虚拟设备，并启动；

▶ 创建虚拟设备

启动虚拟设备

虚拟设备在操作上与真机基本一致，且可模拟各类传感器；



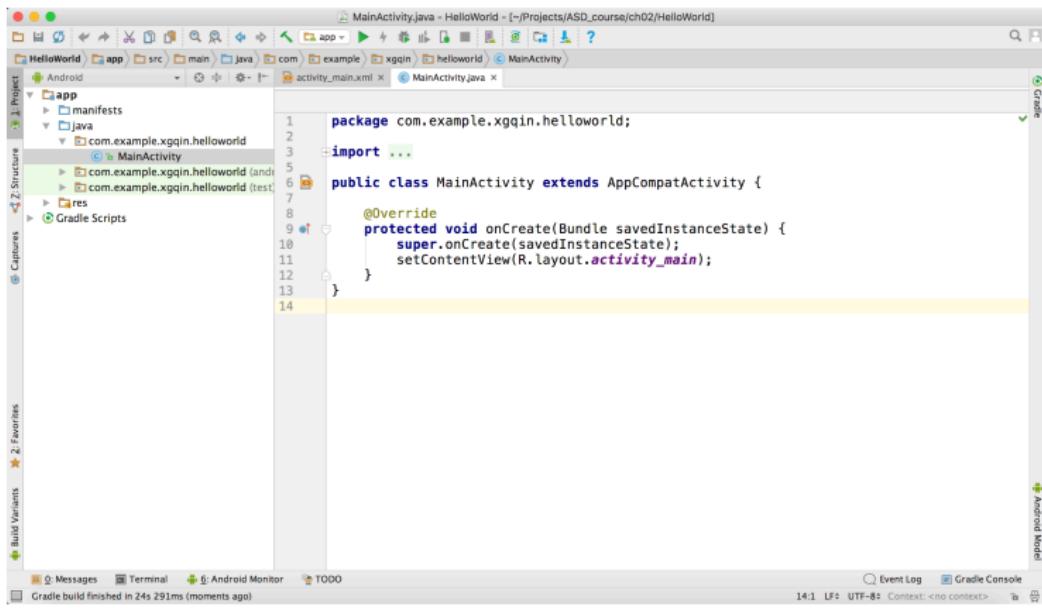
运行 *Hello world*

点击▶图标，Android Studio 对当前工程进行编译，如无错误，将 App 安装至指定的虚拟设备，并启动该 App 运行。



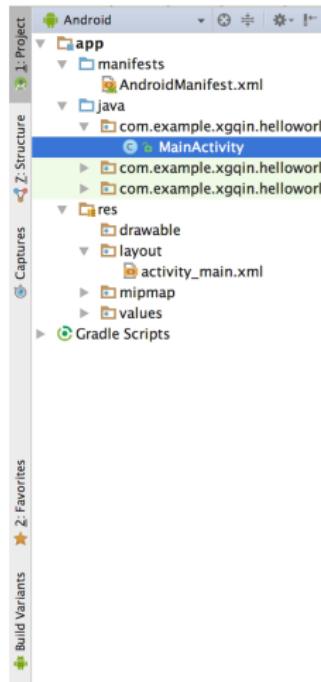
Android 工程与 Project 窗口

在 Android Studio 中，工程包含了进行应用开发所需的所有文件及配置信息（包括：源码、资源文件、测试文件、构建配置等）；



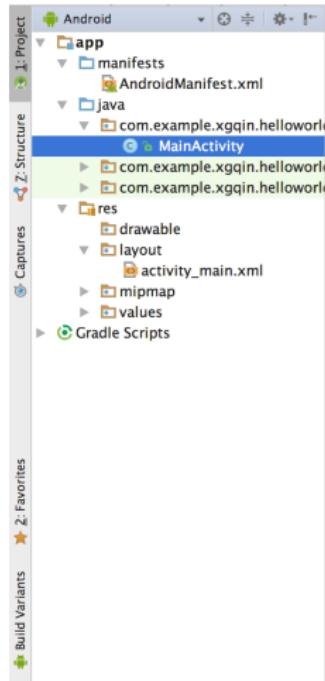
Android 工程结构 — “Android view”

默认情况下，Android Studio 以“Android view”(Android 视角) 形式显示工程文件；



Android 工程结构 — “Android view”

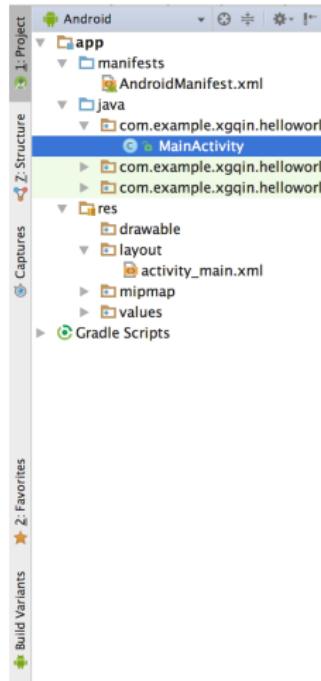
默认情况下，Android Studio 以“Android view”(Android 视角) 形式显示工程文件；



在 Android 视角下，所有的构建文件在“Gradle Scripts”目录树的顶层显示，每一个模块包含如下文件夹：

Android 工程结构 — “Android view”

默认情况下，Android Studio 以“Android view”(Android 视角) 形式显示工程文件；



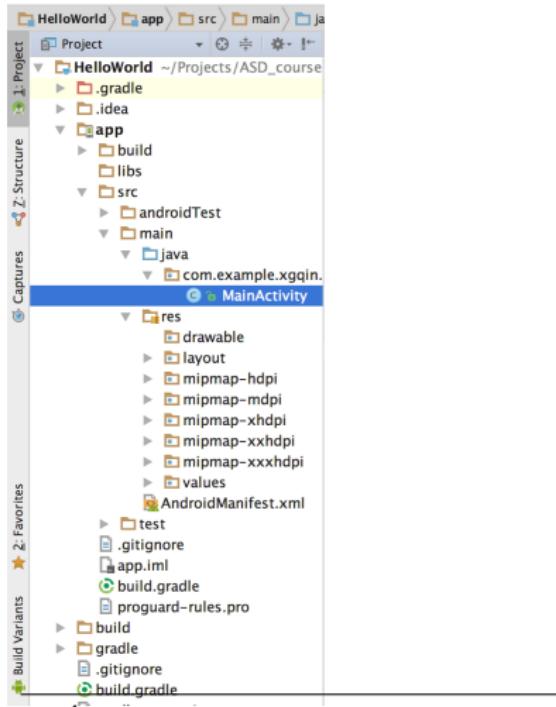
在 Android 视角下，所有的构建文件在“Gradle Scripts”目录树的顶层显示，每一个模块包含如下文件夹：

- **manifests**: 包含 AndroidManifest.xml 文件；
- **java**: 包含 Java 源代码文件以及 JUnit 测试代码；
- **res**: 包含所有非代码资源，例如 XML 布局文件，UI 字符串，位图等；

▶ Android Module

Android 工程结构 — “Project view”

选择 “Project view”(工程视角) 可以查看与工程在本地文件系统对应的目录及文件结构⁴；

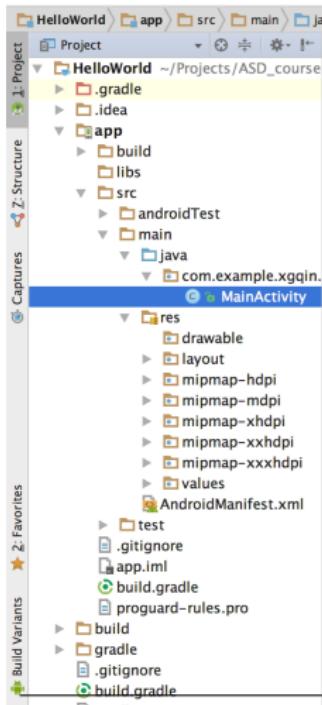


4

<https://developer.android.com/studio/projects/index.html#ProjectStructure>

Android 工程结构 — “Project view”

选择 “Project view”(工程视角) 可以查看与工程在本地文件系统对应的目录及文件结构⁴；



module-name/

● ***build/***

● ***lib/***

● ***src/***

● ***androidTest/***

● ***main/***

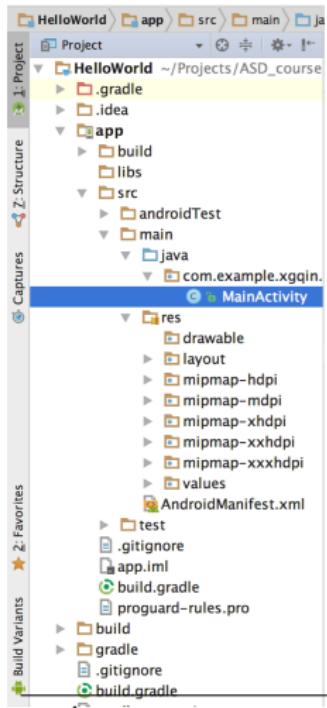
● ***test/***

● ***build.gradle(module)***

build.gradle (project)

Android 工程结构 — “Project view”

选择 “Project view”(工程视角) 可以查看与工程在本地文件系统对应的目录及文件结构⁴；



main/

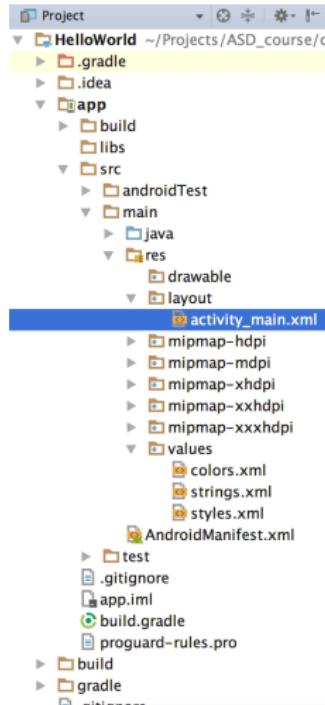
- AndroidManifest.xml
- java/
- jni/
- gen/
- res/
- assets/

4

<https://developer.android.com/studio/projects/index.html#ProjectStructure>

在 AndroidManifest.xml 注册 Activity

app/src/main/AndroidManifest.xml



在 AndroidManifest.xml 注册 Activity

app/src/main/AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.xgqin.helloworld">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="@string/app_name"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/AppTheme">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20
21 </manifest>
```

MainActivity 类定义

app/src/main/java/com.example.xgqin.helloworld/MainActivity.java

```
1 package com.example.xgqin.helloworld;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class MainActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

MainActivity 类定义

app/src/main/java/com.example.xgqin.helloworld/MainActivity.java

```
1 package com.example.xgqin.helloworld;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class MainActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

MainActivity 类中，onCreate(Bundle savedInstanceState)在一个Activity 实例被创建时必须执行的方法；

MainActivity 类定义

app/src/main/java/com.example.xgqin.helloworld/MainActivity.java

```
1 package com.example.xgqin.helloworld;  
2  
3 import android.support.v7.app.AppCompatActivity;  
4 import android.os.Bundle;  
5  
6 public class MainActivity extends AppCompatActivity {  
7  
8     @Override  
9     protected void onCreate(Bundle savedInstanceState) {  
10         super.onCreate(savedInstanceState);  
11         setContentView(R.layout.activity_main);  
12     }  
13 }
```

MainActivity 类中，onCreate(Bundle savedInstanceState)在一个Activity 实例被创建时必须执行的方法；

setContentView(R.layout.activity_main); 给当前 Activity 引入布局 (Layout);

MainActivity 对应的布局文件 — activity_main.xml

app/src/main/res/layout/activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.constraint.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context="com.example.xgqin.helloworld.MainActivity">
9
10    <TextView
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:text="HelloWorld!"
14        app:layout_constraintBottom_toBottomOf="parent"
15        app:layout_constraintLeft_toLeftOf="parent"
16        app:layout_constraintRight_toRightOf="parent"
17        app:layout_constraintTop_toTopOf="parent" />
18
19 </android.support.constraint.ConstraintLayout>
```

改进目标 — 向一个新 Activity 发送消息

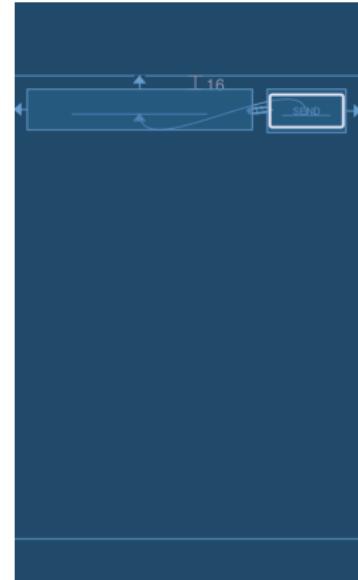
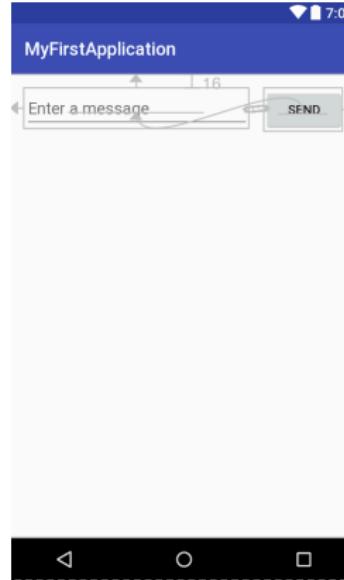


改进目标 — 向一个新 Activity 发送消息

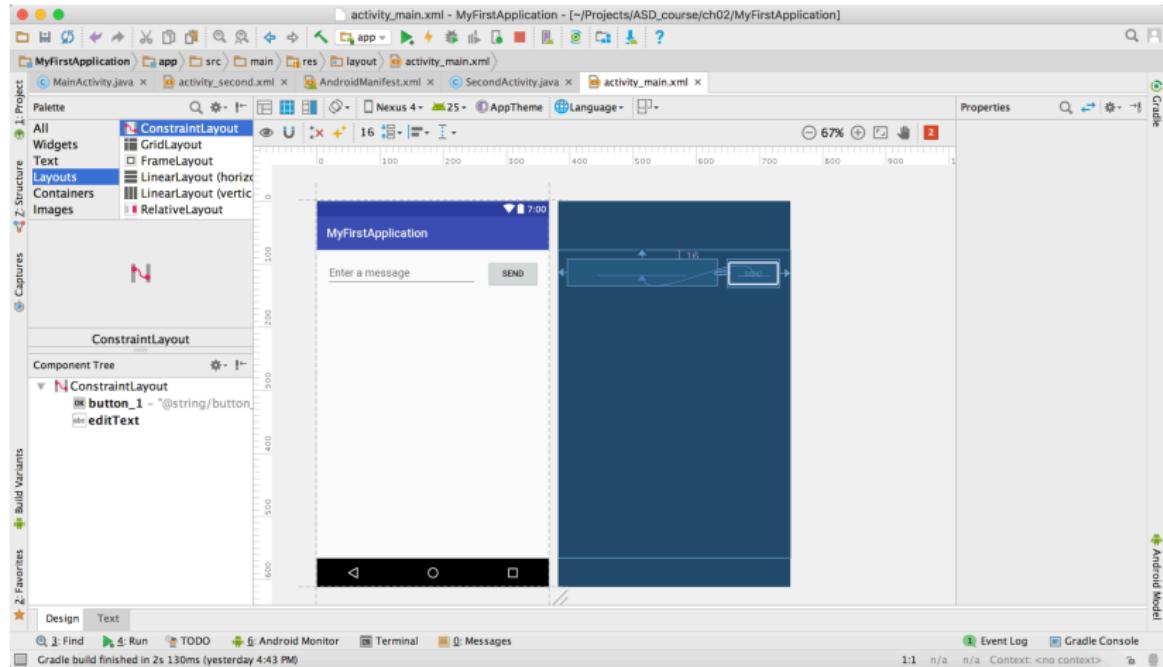


- 新增 Activity，并通过 Intent 及 startActivity () 启动；
- 了解 Layout Editor 基本使用；
- 了解 AndroidManifest.xml 文件作用及基本格式；
- 了解 Intent 的基本用法；

对 MainActivity 进行重新布局



认识 Layout Editor



通过点击 Layout Editor 左下方的“Desgin”与“Text”标签，可以在设计模式 (Desgin Mode) 与文本模式 (Text Mode) 之间进行切换：

Layout Editor 的设计模式与文本模式 |

以下是在文本模式下显示的 activity_main.xml 文件内容：

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.constraint.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context="com.example.xgqin.myfirstapplication.MainActivity">
9
10    <Button
11        android:id="@+id/button_1"
12        android:layout_width="wrap_content"
13        android:layout_height="wrap_content"
14        android:layout_marginLeft="16dp"
15        android:layout_marginRight="16dp"
16        android:onClick="click_me"
17        android:text="@string/button_send"
18        app:layout_constraintBaseline_toBaselineOf="@+id/editText"
```

Layout Editor 的设计模式与文本模式 II

```
19      app:layout_constraintLeft_toRightOf="@+id/editText"
20      app:layout_constraintRight_toRightOf="parent" />
21
22  <EditText
23      android:id="@+id/editText"
24      android:layout_width="0dp"
25      android:layout_height="45dp"
26      android:layout_marginTop="16dp"
27      android:ems="10"
28      android:hint="@string/edit_message"
29      android:inputType="textPersonName"
30      app:layout_constraintLeft_toLeftOf="parent"
31      app:layout_constraintTop_toTopOf="parent"
32      app:layout_constraintRight_toLeftOf="@+id/button_1"
33      android:layout_marginLeft="16dp" />
34
35 </android.support.constraint.ConstraintLayout>
```

Layout Editor 的设计模式与文本模式 III

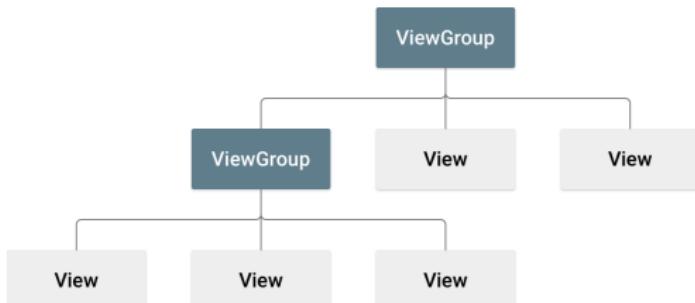
在 `activity_main.xml` 布局文件中，包含如下三个元素：

- ConstraintLayout
- Button
- EditText

这三个元素在布局文件中使用标签进行描述；且在标签中可以指定各元素的属性；

Activity 的 UI 组成 |

Android 应用中的 UI 界面由层次化的布局管理器 (layouts, ViewGroup 对象) 以及窗口部件 (widgets, View 对象) 组成。布局管理器是不可见的容器，用于对其包含的子窗口 (views) 进行布局管理；窗口部件则是例如按钮、文本框之类的 UI 组件。



为 Send 按钮添加单击事件处理函数 |

为 MainActivity类中的“Send”按钮添加单击事件，需要在 MainActivity类的onCreate(Bundle savedInstanceState)方法中编写相应代码；

实现为“Send”按钮添加单击事件通常需要如下几步操作：

- ① 获得指定按钮的实例；通过 View findViewById(`int id`)；
- ② 为该按钮实例添加事件监听函数；通过
`setOnClickListener (OnClickListener l))`；

为 Send 按钮添加单击事件处理函数 II

```
11 public class MainActivity extends AppCompatActivity {  
12     public static final String EXTRA_MESSAGE = "com.example.myfirstApplication.MESSAGE";  
13  
14     @Override  
15     protected void onCreate(Bundle savedInstanceState) {  
16         super.onCreate(savedInstanceState);  
17         setContentView(R.layout.activity_main);  
18  
19         Button button = (Button) findViewById(R.id.button_1);  
20         button.setOnClickListener(new View.OnClickListener() {  
21             @Override  
22             public void onClick(View v) {  
23                 Intent intent = new Intent(MainActivity.this, SecondActivity.class);  
24                 EditText editText = (EditText) findViewById(R.id.editText);  
25  
26                 String message = editText.getText().toString();  
27                 intent.putExtra(EXTRA_MESSAGE, message);  
28  
29                 startActivity(intent);  
30             }  
31         });  
32     }  
33 };
```

为 Send 按钮添加单击事件处理函数 III

```
32      }  
33 }
```

为 Send 按钮添加单击事件处理函数

findViewById 方法获取指定控件实例

```
19     Button button = (Button) findViewById(R.id.button_1);
```

R.id.button_1 表示，指定的 UI 控件 id，其在 activity_main.xml 文件 Button 标签中进行了指定；

为 Send 按钮添加单击事件处理函数

通过 `setOnClickListener(OnClickListener l)` 方法为按钮添加单击事件处理函数。其中 `OnClickListener`⁵ 为接口，该接口中声明了 `onClick` 方法；因此需要对该方法进行重载⁶；

```
19     Button button = (Button) findViewById(R.id.button_1);
20     button.setOnClickListener(new View.OnClickListener() {
21         @Override
22         public void onClick(View v) {
23             Intent intent = new Intent(MainActivity.this, SecondActivity.class);
24             EditText editText = (EditText) findViewById(R.id.editText);
25
26             String message = editText.getText().toString();
27             intent.putExtra(EXTRA_MESSAGE, message);
28
29             startActivity(intent);
30         }
31     });
32 }
```

⁵ <https://developer.android.com/reference/android/view/View.OnClickListener.html>

⁶ 本例使用中给 `setOnClickListener()` 方法传递的是匿名类，其本质就是实现回调；

为 Send 按钮添加单击事件处理函数

使用显示 Intent 启动另外一个 Activity

```
19     Button button = (Button) findViewById(R.id.button_1);
20     button.setOnClickListener(new View.OnClickListener() {
21         @Override
22         public void onClick(View v) {
23             Intent intent = new Intent(MainActivity.this, SecondActivity.class);
24             EditText editText = (EditText) findViewById(R.id.editText);
25
26             String message = editText.getText().toString();
27             intent.putExtra(EXTRA_MESSAGE, message);
28
29             startActivity(intent);

```

为 Send 按钮添加单击事件处理函数

使用显示 Intent 启动另外一个 Activity

```
19     Button button = (Button) findViewById(R.id.button_1);
20     button.setOnClickListener(new View.OnClickListener() {
21         @Override
22         public void onClick(View v) {
23             Intent intent = new Intent(MainActivity.this, SecondActivity.class);
24             EditText editText = (EditText) findViewById(R.id.editText);
25
26             String message = editText.getText().toString();
27             intent.putExtra(EXTRA_MESSAGE, message);
28
29             startActivity(intent);

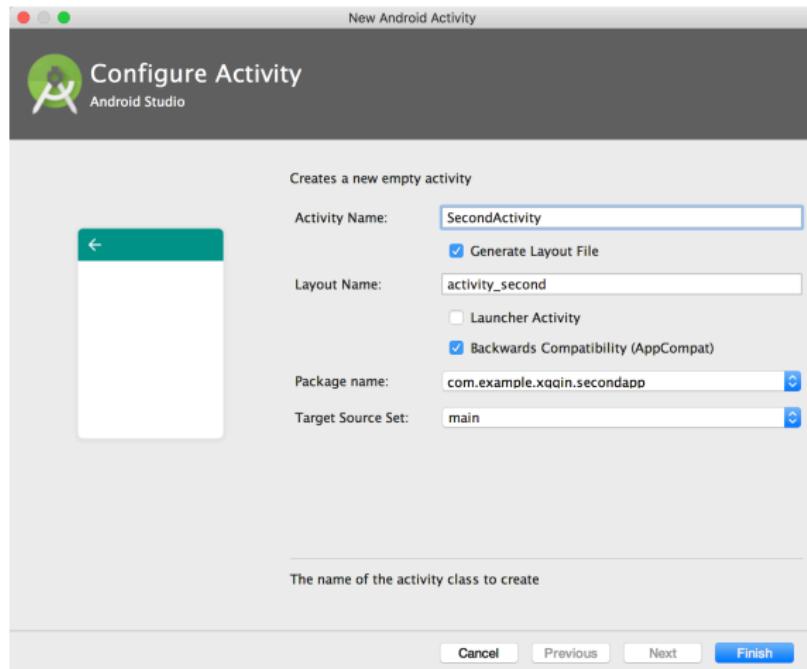
```

Intent 构造函数 Intent(Context packageContext, Class<?> cls) 需要指明两个参数：

- 一个 Context 上下文对象；在此例中传入 MainActivity.this；
- 要显示启动的 Android 组件，这里指明组件的类名 DisplayMessageActivity.class；

添加另一个 Activity

在 Project View 中，右击 [app] -> [New] -> [Activity] -> [Empty Activity] 为工程添加第二个 Activity；



在 SecondActivity 中接收 Intent 传递过来的数据

为 SecondActivity 对应的布局文件新增一个 TextView 控件，并在 SecondActivity 类的 onCreate 方法中获取启动该 Activity 的 Intent 对象；

```
8 public class SecondActivity extends AppCompatActivity {
9
10    @Override
11    protected void onCreate(Bundle savedInstanceState) {
12        super.onCreate(savedInstanceState);
13        setContentView(R.layout.activity_second);
14
15        Intent intent = getIntent();
16        String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
17
18        TextView textView = (TextView) findViewById(R.id.textView);
19        textView.setText(message);
20    }
21 }
```

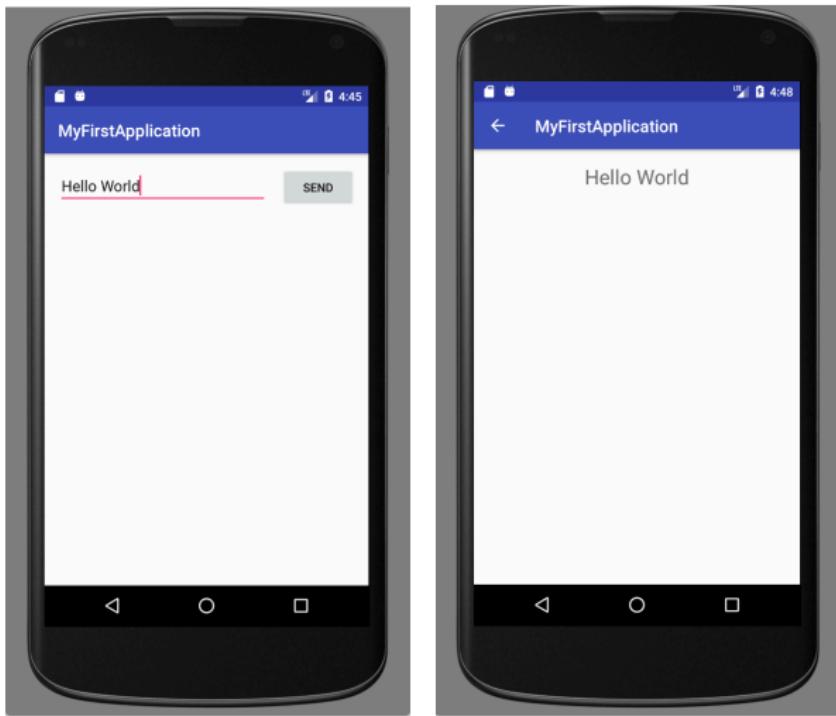
为 SecondActivity 增加顶部返回按钮！

在 Android App 中，对于非主入口的 UI 界面，均需提供返回导航按钮，以便用户能返回到当前界面的父界面（上层界面）中。

这一效果的实现，通常是为用户提供界面顶部返回按钮来实现的。要实现这一操作，仅需要在 `AndroidManifest.xml` 中对应的 `Activity` 标签内通过嵌套 `<meta-data>` 标签进行配置即可：

```
19     <activity android:name=".SecondActivity">
20         <meta-data
21             android:name="android.support.PARENT_ACTIVITY"
22             android:value=".MainActivity"
23         />
24     </activity>
```

App 最终运行效果



AndroidManifest.xml 是什么？

在 Android 应用当中，必须包含一个 **AndroidManifest.xml** 文件（文件名必须相同）；该文件提供了应用运行时，Android 系统所需应用信息。这其中包括：

- 应用包名；应用包名作为应用的唯一标识符；

⁷ 包括各组件对应的类名、各组件提供的功能等描述信息；这类描述信息可为 Android 系统如何启动这类组件提供必要帮助。

AndroidManifest.xml 是什么？

在 Android 应用当中，必须包含一个 **AndroidManifest.xml** 文件（文件名必须相同）；该文件提供了应用运行时，Android 系统所需应用信息。这其中包括：

- 应用包名；应用包名作为应用的唯一标识符；
- 应用所包含的组件⁷，包括构成应用的 Activities、Services、Broadcast Receivers 以及 Content Providers；

⁷ 包括各组件对应的类名、各组件提供的功能等描述信息；这类描述信息可为 Android 系统如何启动这类组件提供必要帮助。

AndroidManifest.xml 是什么？

在 Android 应用当中，必须包含一个 **AndroidManifest.xml** 文件（文件名必须相同）；该文件提供了应用运行时，Android 系统所需应用信息。这其中包括：

- 应用包名；应用包名作为应用的唯一标识符；
- 应用所包含的组件⁷，包括构成应用的 Activities、Services、Broadcast Receivers 以及 Content Providers；
- 决定能运行这个应用包含组件的进程；

⁷ 包括各组件对应的类名、各组件提供的功能等描述信息；这类描述信息可为 Android 系统如何启动这类组件提供必要帮助。

AndroidManifest.xml 是什么？

在 Android 应用当中，必须包含一个 **AndroidManifest.xml** 文件（文件名必须相同）；该文件提供了应用运行时，Android 系统所需应用信息。这其中包括：

- 应用包名；应用包名作为应用的唯一标识符；
- 应用所包含的组件⁷，包括构成应用的 Activities、Services、Broadcast Receivers 以及 Content Providers；
- 决定能运行这个应用包含组件的进程；
- **描述应用在运行受保护 API 时及与其他应用进行交互时所需的权限，以及其他应用与本系统组件进行交互时所必须具备的权限；**

⁷ 包括各组件对应的类名、各组件提供的功能等描述信息；这类描述信息可为 Android 系统如何启动这类组件提供必要帮助。

AndroidManifest.xml 是什么？

在 Android 应用当中，必须包含一个 **AndroidManifest.xml** 文件（文件名必须相同）；该文件提供了应用运行时，Android 系统所需应用信息。这其中包括：

- 应用包名；应用包名作为应用的唯一标识符；
- 应用所包含的组件⁷，包括构成应用的 Activities、Services、Broadcast Receivers 以及 Content Providers；
- 决定能运行这个应用包含组件的进程；
- 描述应用在运行受保护 API 时及与其他应用进行交互时所需的权限，以及其他应用与本系统组件进行交互时所必须具备的权限；
- **指明应用所能支持的最低 API 版本；**

⁷ 包括各组件对应的类名、各组件提供的功能等描述信息；这类描述信息可为 Android 系统如何启动这类组件提供必要帮助。

AndroidManifest.xml 是什么？

在 Android 应用当中，必须包含一个 **AndroidManifest.xml** 文件（文件名必须相同）；该文件提供了应用运行时，Android 系统所需应用信息。这其中包括：

- 应用包名；应用包名作为应用的唯一标识符；
- 应用所包含的组件⁷，包括构成应用的 Activities、Services、Broadcast Receivers 以及 Content Providers；
- 决定能运行这个应用包含组件的进程；
- 描述应用在运行受保护 API 时及与其他应用进行交互时所需的权限，以及其他应用与本系统组件进行交互时所必须具备的权限；
- 指明应用所能支持的最低 API 版本；
- **指明应用所需的链接库等；**

⁷ 包括各组件对应的类名、各组件提供的功能等描述信息；这类描述信息可为 Android 系统如何启动这类组件提供必要帮助。

AndroidManifest.xml 结构 |

以下给出 AndroidManifest.xml 所能包含的子元素：

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest>
3     <uses-permission />
4     <permission />
5     <permission-tree />
6     <permission-group />
7     <instrumentation />
8     <uses-sdk />
9     <uses-configuration />
10    <uses-feature />
11    <supports-screens />
12    <compatible-screens />
13    <supports-gl-texture />
14
15    <application>
16
17        <activity>
18            <intent-filter>
```

AndroidManifest.xml 结构 II

```
19          <action />
20          <category />
21          <data />
22      </intent-filter>
23      <meta-data />
24  </activity>
25
26  <activity-alias>
27      <intent-filter> . . . </intent-filter>
28      <meta-data />
29  </activity-alias>
30
31  <service>
32      <intent-filter> . . . </intent-filter>
33      <meta-data/>
34  </service>
35
36  <receiver>
37      <intent-filter> . . . </intent-filter>
38      <meta-data />
39  </receiver>
40
41  <provider>
```

AndroidManifest.xml 结构 III

```
42      <grant-uri-permission />
43      <meta-data />
44      <path-permission />
45  </provider>
46
47      <uses-library />
48  </application>
49 </manifest>
```

AndroidManifest.xml 文件约束规则

元素约束规则

AndroidManifest.xml 文件 — 元素 (Element) 约束规则：

- 仅 `<manifest>` 及 `<application>` 元素是必须的；且这两个元素只能出现一次；其他元素可出现多次或不出现；

AndroidManifest.xml 文件约束规则

元素约束规则

AndroidManifest.xml 文件 — 元素 (Element) 约束规则：

- 仅 `<manifest>` 及 `<application>` 元素是必须的；且这两个元素只能出现一次；其他元素可出现多次或不出现；
- 元素可以包含其他元素；元素所有的值均以元素属性形式存在，不可以元素字符数据形式存在；

AndroidManifest.xml 文件约束规则

元素约束规则

AndroidManifest.xml 文件 — 元素 (Element) 约束规则：

- 仅 `<manifest>` 及 `<application>` 元素是必须的；且这两个元素只能出现一次；其他元素可出现多次或不出现；
- 元素可以包含其他元素；元素所有的值均以元素属性形式存在，不可以元素字符数据形式存在；
- 同层元素之间是无序的；但以下几条规则是例外：

AndroidManifest.xml 文件约束规则

元素约束规则

AndroidManifest.xml 文件 — 元素 (Element) 约束规则：

- 仅 `<manifest>` 及 `<application>` 元素是必须的；且这两个元素只能出现一次；其他元素可出现多次或不出现；
- 元素可以包含其他元素；元素所有的值均以元素属性形式存在，不可以元素字符数据形式存在；
- 同层元素之间是无序的；但以下几条规则是例外：
 - `<activity-alias>` 元素必须紧跟 `<activity>` 元素，用做前者的别名 (alias)；

AndroidManifest.xml 文件约束规则

元素约束规则

AndroidManifest.xml 文件 — 元素 (Element) 约束规则：

- 仅 `<manifest>` 及 `<application>` 元素是必须的；且这两个元素只能出现一次；其他元素可出现多次或不出现；
- 元素可以包含其他元素；元素所有的值均以元素属性形式存在，不可以元素字符数据形式存在；
- 同层元素之间是无序的；但以下几条规则是例外：
 - `<activity-alias>` 元素必须紧跟 `<activity>` 元素，用做前者的别名 (alias)；
 - `<application>` 元素必须是 `<manifest>` 元素内的最后一个元素；换而言之 `</application>` 闭标签后必须紧跟 `</manifest>` 闭标签；

AndroidManifest.xml 文件约束规则

元素约束规则

AndroidManifest.xml 文件 — 属性 (Attribute) 约束规则：

- 在大多数情况下，元素的属性都是可选的。但是要使元素发挥作用，某些属性必须被指定；以整个 AndroidManifest.xml 为例，当某些属性未指定时，将采用该属性的默认值；

AndroidManifest.xml 文件约束规则

元素约束规则

AndroidManifest.xml 文件 — 属性 (Attribute) 约束规则：

- 在大多数情况下，元素的属性都是可选的。但是要使元素发挥作用，某些属性必须被指定；以整个 AndroidManifest.xml 为例，当某些属性未指定时，将采用该属性的默认值；
- 除了根元素 **<manifest>** 的某些属性外，其他所有属性名均以 *android:* 做为前缀；例如：
android:alwaysRetainTaskState；因为属性前缀都是统一的，因此文档中在通过属性名引用属性时，通常忽略其前缀；

AndroidManifest.xml 文件约束规则

在 AndroidManifest 中声明类名 |

当定义一个组件的子类时，需要在 AndroidManifest 中声明其对应的元素⁸，并通过 *name* 属性指明其类名；*name* 属性必须是包含应用程序完整包名，例如：

⁸ AndroidManifest 文件中大多数元素均对应于 Java 对象，包括 `application` 元素 (`<application>`) 自身在内，以及其主要组件：`activities`(`<activity>`)、`services`(`<service>`)、`broadcast receivers`(`<receiver>`) 以及 `content providers`(`<provider>`)。

AndroidManifest.xml 文件约束规则

在 AndroidManifest 中声明类名 |

当定义一个组件的子类时，需要在 AndroidManifest 中声明其对应的元素⁸，并通过 *name* 属性指明其类名；*name* 属性必须是包含应用程序完整包名，例如：

```
1  <manifest . . . >
2      <application . . . >
3          <service android:name="com.example.project.SecretService" . . . >
4              .
5          </service>
6          .
7      </application>
8  </manifest>
```

⁸ AndroidManifest 文件中大多数元素均对应于 Java 对象，包括 *application* 元素 (`<application>`) 自身在内，以及其主要组件：*activities*(`<activity>`)、*services*(`<service>`)、*broadcast receivers*(`<receiver>`) 以及 *content providers*(`<provider>`)。

AndroidManifest.xml 文件约束规则

在 AndroidManifest 中声明类名 II

如果 *name* 属性的第一个字符是. 时，应用程序的包名⁹将作为该属性的前缀；以下两种声明方式效果相同¹⁰：

```
1 <manifest package="com.example.project" ... >
2   <application ... >
3     <service android:name=".SecretService" ... >
4     </service>
5   </application>
6 </manifest>
```

⁹ 在 <manifest> 元素的 *package* 属性中指明

¹⁰ 当需要启动一个组件时，Android 系统会创建该组件对应的子类实例。如果该子类名未在 AndroidManifest 对

AndroidManifest.xml 文件约束规则

在 AndroidManifest 中声明类名 II

如果 *name* 属性的第一个字符是. 时，应用程序的包名⁹将作为该属性的前缀；以下两种声明方式效果相同¹⁰：

```
1 <manifest package="com.example.project" ... >
2   <application ... >
3     <service android:name=".SecretService" ... >
4     </service>
5   </application>
6 </manifest>
```

```
1 <manifest ... >
2   <application ... >
3     <service android:name="com.example.project.SecretService" ... >
4     </service>
5   </application>
6 </manifest>
```

⁹ 在 <manifest> 元素的 *package* 属性中指明

¹⁰ 当需要启动一个组件时，Android 系统会创建该组件对应的子类实例。如果该子类名未在 AndroidManifest 对

AndroidManifest.xml 文件约束规则

元素声明多个相同属性值

当元素中存在多个属性值时，属性对应的元素应声明多次，而不是将这些多个属性值置于一个元素中。例如，一个 intent filter 可以包括多个 actions：

```
1 <intent-filter ... >
2   <action android:name="android.intent.action.EDIT" />
3   <action android:name="android.intent.action.INSERT" />
4   <action android:name="android.intent.action.DELETE" />
5   ...
6 </intent-filter>
```

AndroidManifest.xml 文件约束规则

资源值

某些元素的属性值对用户而言是可见的，例如 Activity 中的 *label* 与 *icon* 属性；这类属性应该支持本地化，并通过资源 (resource) 或主题 (theme) 进行设置。资源值通常以如下形式进行指定¹¹：

@[package:]type/name

¹¹ 属性值若设定从主题 (theme) 中指定时，其格式为：?@[package:]type/name

¹² 资源或主题的包名必须是 “android” 或你应用的包名

AndroidManifest.xml 文件约束规则

资源值

某些元素的属性值对用户而言是可见的，例如 Activity 中的 *label* 与 *icon* 属性；这类属性应该支持本地化，并通过资源 (resource) 或主题 (theme) 进行设置。资源值通常以如下形式进行指定¹¹：

@[package:]type/name

如果资源与当前组件在同一个包中，则 *package* 名¹²可忽略；*type* 指明资源类型，例如: *string*, *drawable*; *name* 则是指定资源的标识符；例如：

```
<activity android:icon="@drawable/smallPic" ...>
```

¹¹ 属性值若设定从主题 (theme) 中指定时，其格式为：?@[package:]type/name

¹² 资源或主题的包名必须是 “android” 或你应用的包名

AndroidManifest.xml 描述组件功能

Intent filters

一个应用的核心组件，例如活动、服务、广播接收者等都是通过 intent 进行激活。intent 可视为一组用于描述期望 action、以及该 action 上操作数据、执行该 action 的组件分类 (category) 以及其他操作的信息集合 (a bundle of information)。

Android 系统通过筛选适合于相应某一 intent 的组件，并实例化该组件，并将该 intent 对象传递给该组件实例，从而响应该 intent 的操作请求。

应用中的组件通过 *intent filters* (intent 过滤器) 的方式指明其能响应的 intent 类别。因此，Android 在启动某一组件时，必须知晓该组件所能处理及相应的 intent。*intent filters* 在 AndroidManifest.xml 文件中的 **<intent-filter>** 元素中指定。

AndroidManifest.xml 描述元素图标与标签

许多元素可以拥有 *icon* 及 *label* 属性，用于指明可对用户显示的一个小图标以及文字标签¹³。

¹³ 某些元素还可设置 *description* 属性，用于对用户显示长文本。

AndroidManifest.xml 描述元素图标与标签

许多元素可以拥有 *icon* 及 *label* 属性，用于指明可对用户显示的一个小图标以及文字标签¹³。

通常来说，一个元素如果设置了 *icon* 及 *label* 属性，那么该 *icon* 及 *label* 也作为这一元素子元素的默认 *icon* 及 *label* 属性。

¹³ 某些元素还可设置 *description* 属性，用于对用户显示长文本。

AndroidManifest.xml 描述元素图标与标签

许多元素可以拥有 *icon* 及 *label* 属性，用于指明可对用户显示的一个小图标以及文字标签¹³。

通常来说，一个元素如果设置了 *icon* 及 *label* 属性，那么该 *icon* 及 *label* 也作为这一元素子元素的默认 *icon* 及 *label* 属性。

```
1  <application
2      android:allowBackup="true"
3      android:icon="@mipmap/ic_launcher"
4      android:label="@string/app_name"
5      android:roundIcon="@app:mipmap/ic_launcher_round"
6      android:supportsRtl="true"
7      android:theme="@style/AppTheme">
8      <activity android:name=".MainActivity">
9          <intent-filter>
10             <action android:name="android.intent.action.MAIN" />
11             <category android:name="android.intent.category.LAUNCHER" />
12         </intent-filter>
13     </activity>
14 </application>
```

¹³ 某些元素还可设置 *description* 属性，用于对用户显示长文本。

AndroidManifest.xml 描述应用权限 |

权限 (permission) 是一个约束规则 (restriction)，用于限制应用访问某些代码模块以及设备数据。权限主要用于限制应用未经授权访问关键数据及代码，从而导致破坏系统及影响用户体验。

AndroidManifest.xml 描述应用权限 I

权限 (permission) 是一个约束规则 (restriction)，用于限制应用访问某些代码模块以及设备数据。权限主要用于限制应用未经授权访问关键数据及代码，从而导致破坏系统及影响用户体验。

在 Android 系统中，每一项权限有一个唯一的标签进行表示，标签名通常表示被限制的权限。例如：

- `android.permission.CALL_EMERGENCY_NUMBERS`
- `android.permission.READ_OWNER_DATA`
- `android.permission.SET_WALLPAPER`
- `android.permission.DEVICE_POWER`

AndroidManifest.xml 描述应用权限 I

权限 (permission) 是一个约束规则 (restriction)，用于限制应用访问某些代码模块以及设备数据。权限主要用于限制应用未经授权访问关键数据及代码，从而导致破坏系统及影响用户体验。

在 Android 系统中，每一项权限有一个唯一的标签进行表示，标签名通常表示被限制的权限。例如：

- android.permission.CALL_EMERGENCY_NUMBERS
- android.permission.READ_OWNER_DATA
- android.permission.SET_WALLPAPER
- android.permission.DEVICE_POWER

如果应用需要访问受权限保护的功能，其必须在 AndroidManifest.xml 文件的 *<uses-permission>* 元素中声明该权限。

应用也可通过权限对自己的组件功能进行保护。应用可以使用由 Android 系统定义的权限¹⁴或由其他应用定义的权限，亦可定义自己的权限。

应用通过在 `AndroidManifest.xml` 文件的 `<permission>` 元素中定义自己的权限。例如：

¹⁴

<https://developer.android.com/reference/android/Manifest.permission.html>

应用也可通过权限对自己的组件功能进行保护。应用可以使用由 Android 系统定义的权限¹⁴或由其他应用定义的权限，亦可定义自己的权限。

应用通过在 AndroidManifest.xml 文件的 *<permission>* 元素中定义自己的权限。例如：

```
1  <manifest ...>
2      <permission android:name="com.example.project.DEBIT_ACCT" ... />
3      <uses-permission android:name="com.example.project.DEBIT_ACCT" />
4      ...
5      <application ...>
6          <activity android:name="com.example.project.FreneticActivity"
7              android:permission="com.example.project.DEBIT_ACCT"
8              ...>
9          </activity>
10         </application>
11     </manifest>
```

¹⁴

<https://developer.android.com/reference/android/Manifest.permission.html>

什么是 Intent(意图)

Intent 是用于响应其他应用组件请求动作的消息对象 (messaging object)。Intent 为组件间提供通信机制，以下是使用 Intent 的几种方式：

¹⁵ 如果需要从启动活动结束时获取返回结果，可以调用 `startActivityForResult()` 方法，并在自身的活动中重载回调函数 `onActivityResult()`；

¹⁶ 在 Android5.0 以上版本，可以通过 `JobScheduler` 启动服务；

什么是 Intent(意图)

Intent 是用于响应其他应用组件请求动作的消息对象 (messaging object)。Intent 为组件间提供通信机制，以下是使用 Intent 的几种方式：

- 启动活动；通过传递 Intent 对象给 `startActivity ()`¹⁵ 的方式启动一个活动，该 Intent 对象指明了需要启动的互动以及传递给该活动的数据；

¹⁵ 如果需要从启动活动结束时获取返回结果，可以调用 `startActivityForResult ()` 方法，并在自身的活动中重载回调函数 `onActivityResult()`；

¹⁶ 在 Android5.0 以上版本，可以通过 `JobScheduler` 启动服务；

什么是 Intent(意图)

Intent 是用于响应其他应用组件请求动作的消息对象 (messaging object)。Intent 为组件间提供通信机制，以下是使用 Intent 的几种方式：

- 启动活动；通过传递 Intent 对象给 `startActivity ()`¹⁵ 的方式启动一个活动，该 Intent 对象指明了需要启动的互动以及传递给该活动的数据；
- 启动服务；服务是一类在后台运行、无用户交互界面的组件¹⁶。通过传递 Intent 对象给 `startService ()` 的方式启动一个活动；

¹⁵ 如果需要从启动活动结束时获取返回结果，可以调用 `startActivityForResult ()` 方法，并在自身的活动中重载回调函数 `onActivityResult()`；

¹⁶ 在 Android5.0 以上版本，可以通过 `JobScheduler` 启动服务；

什么是 Intent(意图)

Intent 是用于响应其他应用组件请求动作的消息对象 (messaging object)。Intent 为组件间提供通信机制，以下是使用 Intent 的几种方式：

- 启动活动；通过传递 Intent 对象给 `startActivity ()`¹⁵ 的方式启动一个活动，该 Intent 对象指明了需要启动的互动以及传递给该活动的数据；
- 启动服务；服务是一类在后台运行、无用户交互界面的组件¹⁶。通过传递 Intent 对象给 `startService ()` 的方式启动一个活动；
- 发送广播；广播是指应用可以接收到的消息事件。Android 系统会通过发送各种不同广播通知应用相应的事件发生，例如系统启动、设备充电等。**

¹⁵ 如果需要从启动活动结束时获取返回结果，可以调用 `startActivityForResult ()` 方法，并在自身的活动中重载回调函数 `onActivityResult()`；

¹⁶ 在 Android 5.0 以上版本，可以通过 `JobScheduler` 启动服务；

Intent 分类

Intent 可以分为两类：

Intent 分类

Intent 可以分为两类：

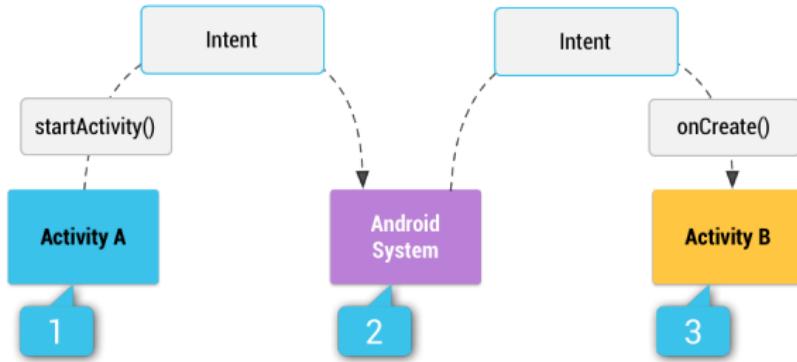
- 显示 intent (Explicit intents)，显示 intent 指明需要启动的组件名字 (the fully-qualified class name)。显示 intent 通常在应用内部自身使用。

Intent 分类

Intent 可以分为两类：

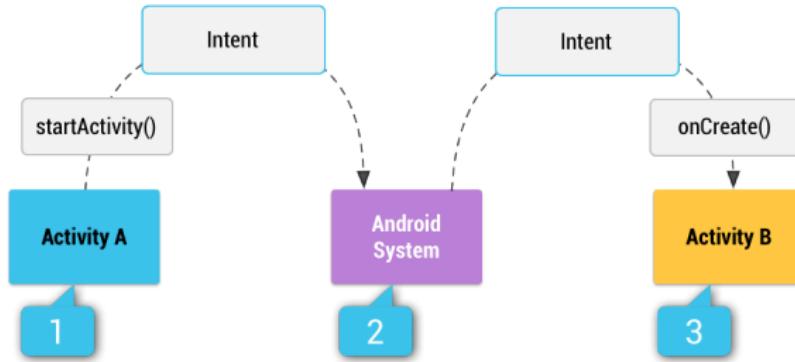
- 显示 intent (Explicit intents)，显示 intent 指明需要启动的组件名字 (the fully-qualified class name)。显示 intent 通常在应用内部自身使用。
- 隐式 intent (Implicit intents)，隐式 intent 不指明启动的组件名，通常以声明需要执行的动作 (action) 代替组件名，至于哪个组件响应该动作，则由 Android 系统进行确定 (intent filter)。

Intent 启动活动的方式



17 通过在本机安装的 App 所提供的 `AndroidManifest.xml` 文件中各组件的 `<intent-filter>` 元素进行匹配。

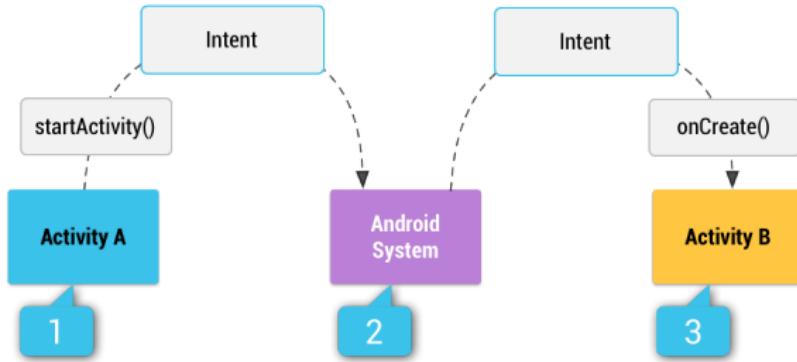
Intent 启动活动的方式



- 通过显示 intent 方式启动活动或服务时，系统会实例化 intent 对象中指定的应用组件；

17 通过在本机安装的 App 所提供的 `AndroidManifest.xml` 文件中各组件的 `<intent-filter>` 元素进行匹配。

Intent 启动活动的方式



- 通过显示 intent 方式启动活动或服务时，系统会实例化 intent 对象中指定的应用组件；
- 通过隐式 intent 方式启动活动或服务时，系统会查找可用于响应该 intent 的组件¹⁷

¹⁷ 通过在本机安装的 App 所提供的 AndroidManifest.xml 文件中各组件的<intent-filter> 元素进行匹配。

如何实例化 Intent 对象

Intent 对象包含 Android 系统决定启动组件所需的信息（例如：详细的组件名，接收 intent 对象的组件 category）以及接收 intent 的组件执行 action 所需的额外数据。

如何实例化 Intent 对象

Intent 对象包含 Android 系统决定启动组件所需的信息（例如：详细的组件名，接收 intent 对象的组件 category）以及接收 intent 的组件执行 action 所需的额外数据。

一个 Intent 对象通常包含以下几类信息：

- Component name;
- Action;
- Data;
- Category;
- Extra;

如何实例化 Intent 对象

Component name

Component name 指明 intent 需启动的组件类名¹⁸；该信息为可选：

¹⁸ 要指定组件名，必须指明组件的完整包名，可以通过setComponent()，setClass()，setClassName()或者Intent的构造函数等形式指定组件名。

如何实例化 Intent 对象

Component name

Component name 指明 intent 需启动的组件类名¹⁸；该信息为可选：

- 如果在 intent 中提供了该信息，则此时使用显示 intent 调用；

¹⁸ 要指定组件名，必须指明组件的完整包名，可以通过setComponent()，setClass()，setClassName()或者Intent的构造函数等形式指定组件名。

如何实例化 Intent 对象

Component name

Component name 指明 intent 需启动的组件类名¹⁸；该信息为可选：

- 如果在 intent 中提供了该信息，则此时使用显示 intent 调用；
- 如果不指定该信息，系统将基于 intent 对象包含的其他信息（例如，action, data, category 等）来决定启动哪一个组件响应，这类方式称为隐式 intent 调用。

¹⁸ 要指定组件名，必须指明组件的完整包名，可以通过setComponent(), setClass(), setClassName()或者Intent的构造函数等形式指定组件名。

如何实例化 Intent 对象

Action

Action 是描述 intent 启动的组件需要执行何操作（例如，view, pick）的字符串。*action* 决定了 intent 其他信息的构建方式，特别是 Data 以及 Extra 两个信息；

¹⁹ 可以在 App 内部自定义 action 以供 intent 使用（或提供给外部 App 调用内部组件时使用）；更通常的做法是使用 Intent 类或其他框架类定义的 action；

²⁰ 可以通过 `setAction()` 方法或 Intent 构造函数指定 action；

如何实例化 Intent 对象

Action

Action 是描述 intent 启动的组件需要执行何操作（例如，view, pick）的字符串。action 决定了 intent 其他信息的构建方式，特别是 Data 以及 Extra 两个信息；

以下是用于启动 activity 所经常使用到的 action^{19,20}：

¹⁹ 可以在 App 内部自定义 action 以供 intent 使用（或提供给外部 App 调用内部组件时使用）；更通常的做法是使用 Intent 类或其他框架类定义的 action；

²⁰ 可以通过 `setAction()` 方法或 Intent 构造函数指定 action；

如何实例化 Intent 对象

Action

Action 是描述 intent 启动的组件需要执行何操作（例如，view, pick）的字符串。action 决定了 intent 其他信息的构建方式，特别是 Data 以及 Extra 两个信息；

以下是用于启动 activity 所经常使用到的 action^{19,20}：

- **ACTION_VIEW** 通过在 `startActivity ()` 传递 intent 时指明该 action，通常用于在新的 Activity 中显示 intent 所传递过去的信息，例如图片，坐标地址等。

¹⁹ 可以在 App 内部自定义 action 以供 intent 使用（或提供给外部 App 调用内部组件时使用）；更通常的做法是使用 Intent 类或其他框架类定义的 action；

²⁰ 可以通过`setAction()`方法或 Intent 构造函数指定 action；

如何实例化 Intent 对象

Action

Action 是描述 intent 启动的组件需要执行何操作（例如，view, pick）的字符串。action 决定了 intent 其他信息的构建方式，特别是 Data 以及 Extra 两个信息；

以下是用于启动 activity 所经常使用到的 action^{19,20}：

- **ACTION_VIEW** 通过在 `startActivity ()` 传递 intent 时指明该 action，通常用于在新的 Activity 中显示 intent 所传递过去的信息，例如图片，坐标地址等。
- **ACTION_SEND** 使用该 action 的 Intent 也被称为分享 intent (share intent)，通过 `startActivity ()` 传递 intent 时指明该 action，通常表示 intent 中有可对其他 App 分享的数据；

¹⁹ 可以在 App 内部自定义 action 以供 intent 使用（或提供给外部 App 调用内部组件时使用）；更通常的做法是使用 Intent 类或其他框架类定义的 action；

²⁰ 可以通过 `setAction()` 方法或 Intent 构造函数指定 action；

如何实例化 Intent 对象

Data

在 Android 系统中，组件之间传递数据的通常采用 URI 进行指明，URI 指明数据的引用或数据的 MIME(Multipurpose Internet Mail Extensions, 是描述消息内容类型的因特网标准²¹)类型。intent 的 action 类型通常隐含了传递的数据类型。

例如：如果 action 指定为**ACTION_EDIT**，则传递的数据应该包含可被编辑的文档的 URI。

²¹ http://www.w3school.com.cn/media/media_mimeref.asp

²² 设置数据的 URI，可以通过setData()方法，设置数据的 MIME 类型，可以通过setType()；如果需要同时设置 URI 及类型，则可通过setDataAndType()方法。

如何实例化 Intent 对象

Data

在 Android 系统中，组件之间传递数据的通常采用 URI 进行指明，URI 指明数据的引用或数据的 MIME(Multipurpose Internet Mail Extensions, 是描述消息内容类型的因特网标准²¹)类型。intent 的 action 类型通常隐含了传递的数据类型。

例如：如果 action 指定为 **ACTION_EDIT**，则传递的数据应该包含可被编辑的文档的 URI。

当创建一个 intent 对象时，除了指明数据的 URI 之外，也应指明数据的类型 (MIME 类型)²²。指明数据的 MIME 类型有助于 Android 系统匹配最佳的组件接收 intent 对象。

²¹ http://www.w3school.com.cn/media/media_mimeref.asp

²² 设置数据的 URI，可以通过 `setData()` 方法，设置数据的 MIME 类型，可以通过 `setType()`；如果需要同时设置 URI 及类型，则可通过 `setDataAndType()` 方法。

如何实例化 Intent 对象

Data

在 Android 系统中，组件之间传递数据的通常采用 URI 进行指明，URI 指明数据的引用或数据的 MIME(Multipurpose Internet Mail Extensions, 是描述消息内容类型的因特网标准²¹)类型。intent 的 action 类型通常隐含了传递的数据类型。

例如：如果 action 指定为 **ACTION_EDIT**，则传递的数据应该包含可被编辑的文档的 URI。

当创建一个 intent 对象时，除了指明数据的 URI 之外，也应指明数据的类型 (MIME 类型)²²。指明数据的 MIME 类型有助于 Android 系统匹配最佳的组件接收 intent 对象。

在某些情况下，数据的 MIME 类型可以由 URI 格式推测出来，特别当数据采用 *content:URI* 形式时。

²¹ http://www.w3school.com.cn/media/media_mimeref.asp

²² 设置数据的 URI，可以通过 `setData()` 方法，设置数据的 MIME 类型，可以通过 `setType()`；如果需要同时设置 URI 及类型，则可通过 `setDataAndType()` 方法。

如何实例化 Intent 对象

Category

在 Intent 对象中，Category 是一个字符串，用于指明所需启动组件的类型（分类）²³。Intent 对象可以指明多个 category，不过在通常情况下，category 不是必须的，以下是常见的 category 类型：

²³ 可以通过 addCategory() 为 intent 添加 Category

如何实例化 Intent 对象

Category

在 Intent 对象中，Category 是一个字符串，用于指明所需启动组件的类型 (分类)²³。Intent 对象可以指明多个 category，不过在通常情况下，category 不是必须的，以下是常见的 category 类型：

- **CATEGORY_BROWSABLE** 该 Category 所指定的 Activity 可以启动浏览器用于显示由 Data 所指定的超链接地址；
- **CATEGORY_LAUNCHER** 该 Category 所指定的 Activity 是任务 (task) 的初始 Activity 并且作为系统中应用启动 Activity(application launcher)；

²³ 可以通过addCategory()为intent添加Category

如何实例化 Intent 对象

Extras

此外，Intent 对象还可携带额外的信息传递给所响应的组件，这些额外信息不影响 Android 系统启动应用组件的策略。

Extra 是 Android 系统中定义的一类键值对 (key-value pair) 类型，可通过 Intent 对象传递给所响应的组件。

²⁴ <https://developer.android.com/reference/android/os/Bundle.html>

²⁵ <https://developer.android.com/guide/components/activities/parcelables-and-bundles.html#sdbp>



如何实例化 Intent 对象

Extras

此外，Intent 对象还可携带额外的信息传递给所响应的组件，这些额外信息不影响 Android 系统启动应用组件的策略。

Extra 是 Android 系统中定义的一类键值对 (key-value pair) 类型，可通过 Intent 对象传递给所响应的组件。

putExtra() 可用于向 Intent 对象添加 Extra 的方法，其接收两个参数，第一个为键名、第二个为值。

也可通过构造一个 Bundle²⁴，²⁵ 对象的方式将所有的 Extra 数据封装在该对象内部，并通过 putExtras() 方法添加至 Intent 对象中。

Intent 类内部定义了众多以 *EXTRA*_ 开头的字符串常量，用于标准化数据类型。用户也可定义自己的 Extra 键。

²⁴ <https://developer.android.com/reference/android/os/Bundle.html>

²⁵ <https://developer.android.com/guide/components/activities/parcelables-and-bundles.html#sdbp>



如何实例化 Intent 对象

Flags

Flag 可以视为 Intent 对象的原数据 (metadata)。Flag 用于指明 Android 系统启动指定的 Activity 的方式 (Activity 属于哪一个 task)。

如何实例化 Intent 对象

显示 Intent 举例

显示 Intent 指通过 Intent 启动指定的应用组件，通常是应用内部的 Activity 或 Service²⁶。要创建显示 Intent，需要在 Intent 对象中指定所需启动的组件名即可（其他 Intent 属性都是可选）。

²⁶

<https://developer.android.com/guide/components/services.html>

如何实例化 Intent 对象

显示 Intent 举例

显示 Intent 指通过 Intent 启动指定的应用组件，通常是应用内部的 Activity 或 Service²⁶。要创建显示 Intent，需要在 Intent 对象中指定所需启动的组件名即可（其他 Intent 属性都是可选）。

例如，应用内部定义了一个名为 *DownloadService* 的 Service 用于从 Web 下载指定文件，则可以通过以下代码实现启动该 Service：

```
1  /* Executed in an Activity, so 'this' is the Context
2   * The fileUrl is a string URL, such as "http://www.example.com/image.png"
3   */
4  Intent downloadIntent = new Intent(this, DownloadService.class);
5  downloadIntent.setData(Uri.parse(fileUrl));
6
7  startService(downloadIntent);
```

²⁶

<https://developer.android.com/guide/components/services.html>

如何实例化 Intent 对象

隐式 Intent 举例

隐式 Intent 通过指定 action 的方式调用安装在本机的其他应用组件。当应用本身无法实现某一 action 时，通过隐式 Intent 可以很好的扩展应用的功能。

如何实例化 Intent 对象

隐式 Intent 举例

隐式 Intent 通过指定 action 的方式调用安装在本机的其他应用组件。当应用本身无法实现某一 action 时，通过隐式 Intent 可以很好的扩展应用的功能。

如果系统无法找到其他应用组件或由于权限限制的原因无法响应处理你的 Intent，`startActivity ()`调用失败，从而造成应用崩溃。可以通过如下形式解决该问题：

如何实例化 Intent 对象

隐式 Intent 举例

隐式 Intent 通过指定 action 的方式调用安装在本机的其他应用组件。当应用本身无法实现某一 action 时，通过隐式 Intent 可以很好的扩展应用的功能。

如果系统无法找到其他应用组件或由于权限限制的原因无法响应处理你的 Intent， startActivity () 调用失败，从而造成应用崩溃。可以通过如下形式解决该问题：

```
1 // Create the text message with a string
2 Intent sendIntent = new Intent();
3 sendIntent.setAction(Intent.ACTION_SEND);
4 sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
5 sendIntent.setType("text/plain");
6
7 // Verify that the intent will resolve to an activity
8 if (sendIntent.resolveActivity(getApplicationContext()) != null) {
9     startActivity(sendIntent);
10 }
```

定义组件所能接收的隐式 Intent

通过 `AndroidManifest.xml` 文件对组件添加 `<intent-filter>` 元素，定义组件所能接收的隐式 Intent。

每一个 `<intent filter>` 指明了基于该 filter 时组件所能接收的 Intent(通过与 Intent 的 action, data, category 进行匹配)。

定义组件所能接收的隐式 Intent

通过 `AndroidManifest.xml` 文件对组件添加 `<intent-filter>` 元素，定义组件所能接收的隐式 Intent。

每一个 `<intent-filter>` 指明了基于该 filter 时组件所能接收的 Intent(通过与 Intent 的 action, data, category 进行匹配)。

每一个 `<intent-filter>` 嵌套在对应的应用组件中，在 `<intent-filter>` 中会通过以下一个或多个元素来指明组件所能接收的 Intent：



定义组件所能接收的隐式 Intent

通过 `AndroidManifest.xml` 文件对组件添加 `<intent-filter>` 元素，定义组件所能接收的隐式 Intent。

每一个 `<intent-filter>` 指明了基于该 filter 时组件所能接收的 Intent(通过与 Intent 的 `action`, `data`, `category` 进行匹配)。

每一个 `<intent-filter>` 嵌套在对应的应用组件中，在 `<intent-filter>` 中会通过以下一个或多个元素来指明组件所能接收的 Intent:

- `<action>` 指明可接受的 Intent 的 `action`, `action` 的名字由 `name` 属性指定。



定义组件所能接收的隐式 Intent

通过 `AndroidManifest.xml` 文件对组件添加 `<intent-filter>` 元素，定义组件所能接收的隐式 Intent。

每一个 `<intent-filter>` 指明了基于该 filter 时组件所能接收的 Intent(通过与 Intent 的 `action`, `data`, `category` 进行匹配)。

每一个 `<intent-filter>` 嵌套在对应的应用组件中，在 `<intent-filter>` 中会通过以下一个或多个元素来指明组件所能接收的 Intent：

- `<action>` 指明可接受的 Intent 的 `action`, `action` 的名字由 `name` 属性指定。
- `<data>` 指明可接受的数据类型，可以通过多个属性约束数据的 URI 各字段；



定义组件所能接收的隐式 Intent

通过 `AndroidManifest.xml` 文件对组件添加 `<intent-filter>` 元素，定义组件所能接收的隐式 Intent。

每一个 `<intent-filter>` 指明了基于该 filter 时组件所能接收的 Intent(通过与 Intent 的 `action`, `data`, `category` 进行匹配)。

每一个 `<intent-filter>` 嵌套在对应的应用组件中，在 `<intent-filter>` 中会通过以下一个或多个元素来指明组件所能接收的 Intent：

- `<action>` 指明可接受的 Intent 的 `action`, `action` 的名字由 `name` 属性指定。
- `<data>` 指明可接受的数据类型，可以通过多个属性约束数据的 URI 各字段；
- `<category>` 指明可接受的 Intent 的 Category, Category 的值由 `name` 属性指定²⁷；

定义组件所能接收的隐式 Intent

```
1 <activity android:name="ShareActivity">
2   <intent-filter>
3     <action android:name="android.intent.action.SEND"/>
4     <category android:name="android.intent.category.DEFAULT"/>
5     <data android:mimeType="text/plain"/>
6   </intent-filter>
7 </activity>
```

如果一个组件可以处理多个类型的 Intent，可以在该组件的 `<intent-filter>` 元素指定多个 `<action>`, `<data>`, `<category>` 或者定义多个 `<intent-filter>`。

定义组件所能接收的隐式 Intent

```
1  <activity android:name="MainActivity">
2      <!-- This activity is the main entry, should appear in app launcher -->
3      <intent-filter>
4          <action android:name="android.intent.action.MAIN" />
5          <category android:name="android.intent.category.LAUNCHER" />
6      </intent-filter>
7  </activity>
8
9  <activity android:name="ShareActivity">
10     <!-- This activity handles "SEND" actions with text data -->
11     <intent-filter>
12         <action android:name="android.intent.action.SEND"/>
13         <category android:name="android.intent.category.DEFAULT"/>
14         <data android:mimeType="text/plain"/>
15     </intent-filter>
16     <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
17     <intent-filter>
18         <action android:name="android.intent.action.SEND"/>
19         <action android:name="android.intent.action.SEND_MULTIPLE"/>
20         <category android:name="android.intent.category.DEFAULT"/>
21         <data android:mimeType="application/vnd.google.panorama360+jpg"/>
22         <data android:mimeType="image/*"/>
23         <data android:mimeType="video/*"/>
24     </intent-filter>
25 </activity>
```

Developer Workflow Basics

开发 Android 应用的流程与其他系统或应用流程类似，以下是在进行 Android 应用开发时所经历的各流程²⁸：



Developer Workflow Basics

开发 Android 应用的流程与其他系统或应用流程类似，以下是在进行 Android 应用开发时所经历的各流程²⁸：



- ① 新建、配置工程；
- ② 编写代码、设计 UI、创建资源、应用程序本地化；
- ③ 构建、配置构建信息等；
- ④ 调试、调优、构建测试代码；
- ⑤ App 发布；

DPI 的概念

Android 设备之间除了屏幕尺寸不同之外，屏幕的像素密度 (DPI, Dots Per Inch) 也不尽相同。

Definition (DPI, Dots Per Inch)

像素密度，用于描述每英寸像素点数量。

设备按照像素密度分类可以分为²⁹：

- LDPI, 120DPI
- MDPI, 160DPI
- HDPI, 240DPI
- XHDPI, 360DPI
- XXHDPI, 480DPI

²⁹ <http://www.zcool.com.cn/article/ZNjI3NDQ=.html>

关于 Module 的一点知识

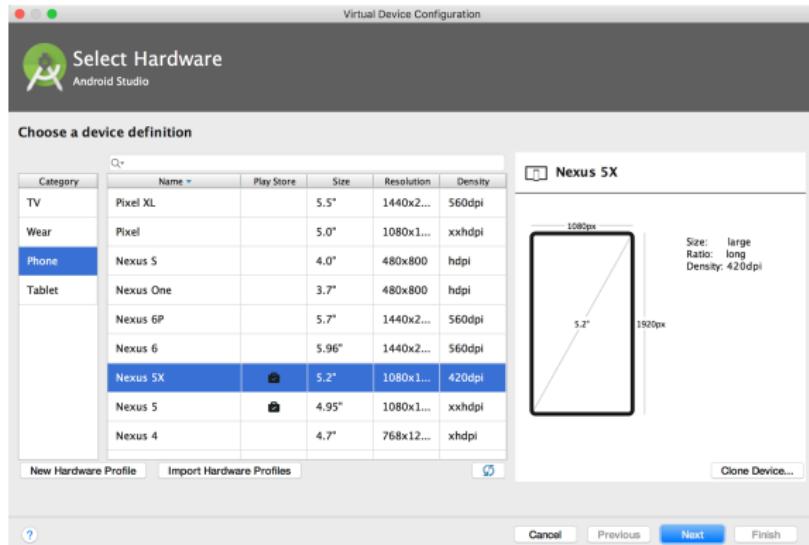
每个 Android Studio 工程包含一个或多个模块 (module)，模块包含特定的源代码文件及资源文件；

在 Android Studio 中，模块分为三类：

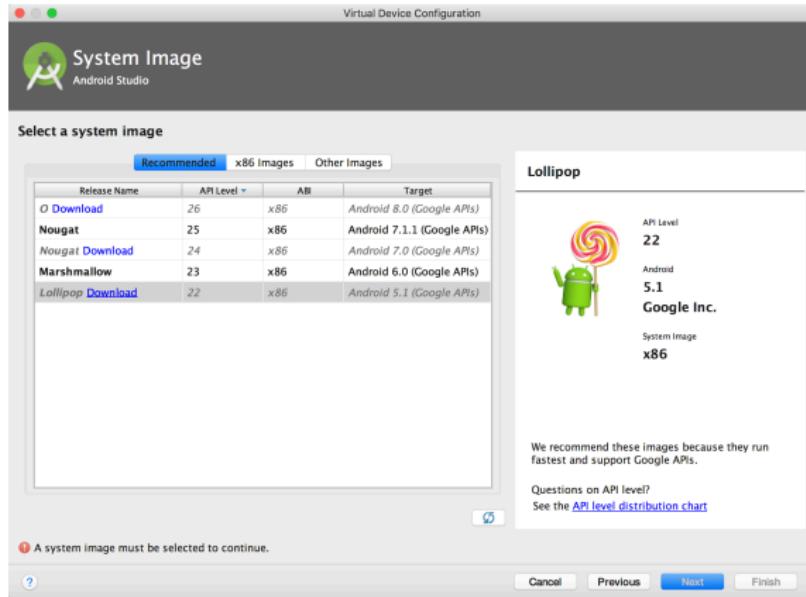
- ① Android app modules
- ② Library modules
- ③ Google App Engine modules

◀ Back

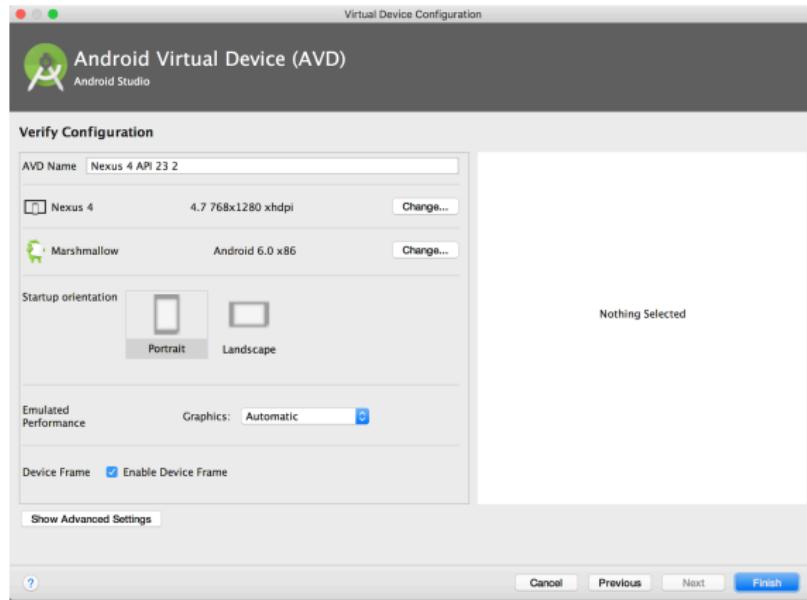
选择虚拟设备对标的硬件



选择 Android 系统版本及镜像



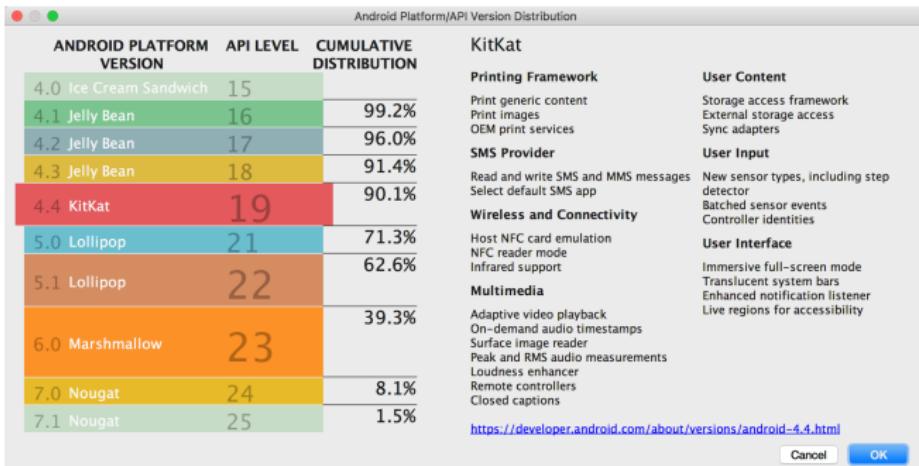
配置虚拟设备信息



◀ 返回

Help me choose

Help me choose 显示最新的 Android 各版本对应的装机率及每个版本的特性，可以帮助开发者确定适合的 Android SDK 进行应用程序开发。



◀ 返回