让你的应用完美适配平板

江江安卓 郭霖 2022-08-05 08:00 发表于江苏

/ 今日科技快讯 /

近日有博主曝光了一条奔驰方面发过来的短信，其中称他的车辆远程启动功能即将到期，而近期给出了400元的优惠，原本1298元三年的远程启动功能，现在仅需898元，一年期的只需498元。买车时就已经为远程启动功能付过费了，现在使用竟然还要交钱。

/ 作者简介 /

大家周五早上好，明天就是周末了，我们下周再见！

本篇文章转自Zhujiang的博客，文章主要分享了平板的适配，相信会对大家有所帮助！

原文地址：

> https://juejin.cn/post/7125250156194168869

/ 前言 /

其实标题有点吹牛逼了，谁也不敢说能完美适配平板，只能说尽力去做，包括显示和使用的各个方面尽力去做，才有可能在更多的平板设备上更加完美的运行起来，因为安卓的设备实在是太多了，之前手机在卷，现在平板也一样在卷。。。

/ 屏幕适配 /

首先来看下最直观的屏幕适配吧，毕竟是直接展示给用户看的嘛！

今日头条的方法

其实说起适配大家首先想到的大名鼎鼎的今日头条的屏幕适配方案，但是！凡事就怕但是！

今日头条适配方案原理在于通过公式 density = 设备真实宽度(单位px)/设计图总宽度(单位dp)，在确保设计图总宽度(单位dp)一定时，通过修改 density 值，确保所有不同尺寸分辨率设备计算出的真实宽度值正好是屏幕宽度。

如果啊，我是说如果！如果所有的安卓设备都是手机的话这不就是完美的解决方案嘛！但事实往往不尽人意，还有 Pad 。。。Pad 和手机完完全全是两码事，屏幕大小差异太大不说，而且 Pad 的常态是横屏，但市面上大多数应用都限制死了竖屏操作，导致应用在 Pad 上根本无法使用或者使用效果特别差！

大家可以看下一些主流应用在 Pad 上的显示效果：

其实不止这一个应用，很多都显示地不尽人意，那么应该如何同时适配好手机和 Pad 呢？

简单粗暴的方法

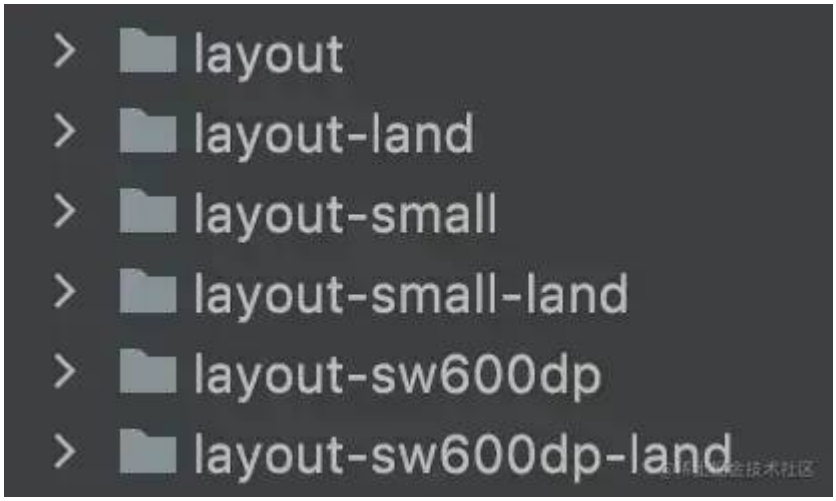
什么方法呢？很简单，直接做两个应用！一个适配手机，另外一个适配 Pad，例如：爱某艺、央某影音、哔哩某哩等等。但是要注意，手机上你可以不适配横竖屏，但是 Pad 上就显得尤为重要了，切换横竖屏就意味着要重走生命周期，重走生命周期就意味着数据都得保存好，数据保存不好就意味着有 bug。。。。

这种方法是土豪做的，一般小公司支撑不住，大家可以量力而行😂😂😂。

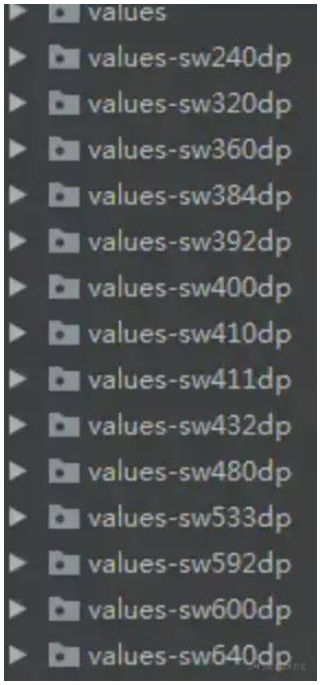
下面来欣赏下哔哩某哩在 Pad 上的显示效果：

普通应用的方法

普通应用其实使用这种方法就可以，什么方法呢？也很简单，就是通过限定符进行限定，编写多套布局来进行适配，类似于下图这样：



我一般限定 Pad 布局使用的是 sw600dp，目前在遇到的 Pad 中显示都还算正常，没有出现太大问题。这种情况由于已经是多套布局，所以也可以使用 **dp适配方案** ，如下图：



具体怎么生成这个的话直接在 as 中搜索 ScreenMatch 插件即可。

使用 Jetpack WindowManager 嵌入 activity

这其实也是 Android 12L 及 Android 13 中推出的新功能，可以利用 activity 嵌入功能，一次显示多个 activity（例如，在列表-详情模式下），以便充分利用大屏设备的额外显示区域，并且只需对应用进行少量重构，甚至无需重构。

更新旧版代码库以支持大屏幕可能需要耗费大量人力和时间，使用 fragment 将基于 activity 的应用转换为多窗格布局需要进行重大重构，可以通过创建 XML 配置文件或进行 Jetpack WindowManager API 调用，确定应用如何显示其 activity（并排或堆叠），系统处理其余的工作，根据创建的配置确定呈现方式。

当然现在很多项目已经在尝试单 Activity 的架构进行开发了，但如果应用使用多个 activity，就可以通过嵌入 activity 来适配 Pad 了。

具体需要怎么做呢?

1. 将 WindowManager 库依赖项添加到 build.gradle 文件中：

```
implementation("androidx.window:window:1.0.0-beta03")
```

2. 创建一个具有以下用途的资源文件：

- 定义应使用过滤器拆分哪些 activity
- 为共享分屏的所有 activity 配置分屏选项
- 指定绝不应放置在分屏中的 activity

例如：

```
<!-- The split configuration for activities. -->
<resources
  xmlns:window="http://schemas.android.com/apk/res-auto">

  <!-- Automatically split the following activity pairs. -->
  <SplitPairRule
    window:splitRatio="0.3"
    window:splitMinWidth="600dp"
    window:finishPrimaryWithSecondary="true"
    window:finishSecondaryWithPrimary="true">
    <SplitPairFilter
      window:primaryActivityName=".SplitActivityList"
      window:secondaryActivityName=".SplitActivityDetail"/>
    <SplitPairFilter
      window:primaryActivityName="*"
      window:secondaryActivityName="*/*"
      window:secondaryActivityAction="android.intent.action.VIEW"/>
  </SplitPairRule>
```

```
    <!-- Automatically launch a placeholder for the list activity. -->
    <SplitPlaceholderRule
        window:placeholderActivityName=".SplitActivityListPlaceholder"
        window:splitRatio="0.3"
        window:splitMinWidth="600dp">
        <ActivityFilter
            window:activityName=".SplitActivityList"/>
    </SplitPlaceholderRule>

</resources>
```

3. 将规则定义通知库。

在本例中，我们使用 Jetpack Startup 库在加载应用的其他组件和启动 activity 之前执行初始化。如需启用启动功能，在应用的 build 文件中添加库依赖项：

```
implementation("androidx.startup:startup-runtime:1.1.0")
```

并在应用清单中添加以下条目：

```
<!-- AndroidManifest.xml -->

<provider android:name="androidx.startup.InitializationProvider"
    android:authorities="${applicationId}.androidx-startup"
    android:exported="false"
    tools:node="merge">
    <!-- This entry makes ExampleWindowInitializer discoverable. -->
    <meta-data  android:name="androidx.window.sample.embedding.ExampleWindowInitializer"
        android:value="androidx.startup" />
</provider>
```

4. 最后，添加初始化程序类实现。

通过将包含定义 (main_split_config) 的 xml 资源文件的 ID 提供给 SplitController.initialize() 来设置规则：

```
class ExampleWindowInitializer extends Initializer<SplitController> {
  @Override
  SplitController create(Context context) {
    SplitController.initialize(context, R.xml.main_split_config);
    return SplitController.getInstance(context);
  }
```

```
  @Override
  List<Class<? extends Initializer<?>>> dependencies() {
    return emptyList();
  }
}
```

屏幕适配大概说到这里，大家可以根据需求和公司人力状况来判断使用哪种方案，条件允许的话维护两个应用的显示效果肯定是最好的，不允许的话剩下几种方案都可以进行选择。

/ 输入兼容性 /

在手机上，用户一般都只会使用手指在屏幕上进行操作，一些特殊的设备会给手机配备手写笔，但由于数量较少基本可以忽略。但是在 Pad 上，用户虽然也会用手机在屏幕上进行操作，但会更频繁地使用键盘、鼠标、触控板、触控笔或游戏手柄与应用互动，这个时候应用的输入兼容性就显得尤为重要！

键盘处理

对于 EditText 等屏幕虚拟键盘处理的文字输入，应用应在大屏幕设备上按预期运行，而无需执行额外操作。但对于系统无法预料的按键，应用需要自行处理相应的行为。

- **普通按键**

比如聊天应用使用 Enter 键发送消息，媒体应用使用空格键开始和停止播放，游戏使用 W、A、S 和 D 键控制移动，等等，这种情况下需要重写 onKeyUp 方法：

```
override fun onKeyUp(keyCode: Int, event: KeyEvent): Boolean {
  return when (keyCode) {
    KeyEvent.KEYCODE_ENTER -> {
      // 回车
      true
    }
    KeyEvent.KEYCODE_SPACE -> {
      // 空格
      true
    }
    else -> super.onKeyUp(keyCode, event)
  }
}
```

在 KeyEvent 类中定义了键盘上的所有操作，由于篇幅原因就不再一一进行列举，大家可以直接去看源码。

其实这里也可以重写 onKeyDown 方法来进行处理按键的事件，但当用户松开键时，会发生 onKeyUp 事件。使用此回调可防止在用户缓慢地按住或松开某个键时应用需要处理多个 onKeyDown 事件。如果游戏和应用想要知道用户何时按了键或预计用户会按住键盘按键，可以查找 onKeyDown() 事件并自行处理重复的 onKeyDown 事件。

注意：根据应用的需求，针对整个 Activity 替换 onKeyUp() 通常可提供所需的行为。如果需要，可以改为向特定的视图添加 onKeyListener。例如，为了只有用户在聊天框中输入消息时才实现发送功能，应用可能只在特定的 EditText（而不是 Activity）中监听 Enter 键。

- **快捷键**

使用硬件键盘时，用户希望实现基于 Ctrl、Alt 和 Shift 的常见快捷键。如果应用不实现这些快捷键，用户可能会觉得应用使用起来不顺手，比如一些常用的快捷键包括 Ctrl + S（保存）、Ctrl + Z（撤消）和 Ctrl + Shift + Z（重做）等等。

要使用快捷键的话可以重写 dispatchKeyShortcutEvent ：

```
override fun dispatchKeyShortcutEvent(event: KeyEvent): Boolean {
    return when (event.keyCode) {
        KeyEvent.KEYCODE_Z -> {
            if (event.isCtrlPressed) {
                if (event.isShiftPressed) {
                    redoLastAction() // Ctrl+Shift+Z pressed
                    true
                } else {
                    undoLastAction() // Ctrl+Z pressed
                    true
                }
            }
        }
        else -> {
            return super.dispatchKeyShortcutEvent(event)
        }
    }
}
```

可以看到上面代码使用到了 isCtrlPressed 和 isShiftPressed，这都是 keyEvent 中的方法，我们可以直接调用来获取当前是否按住 Ctrl 、Shift 或 alt 键，当然还有别的很多键按住的判断，大家可以去翻源码看看。

触控笔的处理

触控笔目前基本成为了 Pad 的标配，不管是否有用，是否需要，每个 Pad 厂家都会出一个触控笔（大部分是模仿某果）。

触控笔事件通过 View.onTouchEvent() 或 View.onGenericMotionEvent() 被报告为触摸屏事件，并且包含返回类型为 SOURCE_STYLUS 的 MotionEvent.getSource()。

MotionEvent 还包含其他数据：

- MotionEvent.getToolType() 将返回 TOOL_TYPE_FINGER、TOOL_TYPE_STYLUS 或 TOOL_TYPE_ERASER，具体取决于与表面接触的工具
- MotionEvent.getPressure() 将报告施加到触控笔的物理压力（需要触控笔支持）
- MotionEvent.getAxisValue() 与 MotionEvent.AXIS_TILT 和 MotionEvent.AXIS_ORIENTATION 一起使用，可用于读取触控笔的物理倾斜度和方向（也需要触控笔支持）

- **历史点**

Android 会对输入事件进行批处理，并且每帧传送一次。触控笔可以按比显示屏高得多的频率来报告事件。创建绘图应用时，需要使用 getHistorical API 检查最近可能发生的事件：

- MotionEvent.getHistoricalX()
- MotionEvent.getHistoricalY()
- MotionEvent.getHistoricalPressure()
- MotionEvent.getHistoricalAxisValue()

- **防止手掌误触**

大多数 Pad 都会尝试识别出用户何时将手掌放到了触摸屏上，但系统并不总是能够做到这一点，有时可能会在操作系统识别出手掌误触之前向应用报告了触摸事件。在这种情况下，系统会通过报告 ACTION_CANCEL 事件来取消触摸，这个时候应用应撤消由这些触摸引起的所有交互。

注意：如需减少绘图和手写应用中无关的手掌和手指误触事件，一种方法是提供相应的界面设置，用于停用通过触摸绘图的功能，在这种模式下仅使用触控笔事件来绘图。

鼠标和触控板支持

鼠标或触控板在电脑上我们使用的太多了，一般有左边按钮点击、右边按钮点击、悬停以及拖拽，下面咱们来分别看看吧。

- **点击**

点击分为左键点击和右键点击，左键点击就是普通按下事件，就不细说了，这里主要来看下右键点击。

右键点击会使应用显示上下文菜单的所有操作（如轻触并按住列表项）也应该对右键点击事件作出反应。为了处理右键点击事件，应用应注册 View.OnContextClickListener。

```
View.setOnContextClickListener {
  showContextMenu()
  true
}
```

注意：如果已使用 Activity.registerForContextMenu() 为上下文菜单注册的所有视图都应自动支持轻触并按住和右键点击，而无需注册上下文点击监听器。

- **悬停**

开发者可以通过处理悬停事件，使其应用布局更美观且更易于使用。对于自定义视图来说尤其如此。这方面最常见的两个示例如下：

  - 通过改变鼠标指针图标，向用户表明某个元素是否具有交互行为，如可点击或可修改

- 当指针悬停在大型列表或网格中的项目上时，向这些项目添加视觉反馈

```
View.setOnHoverListener { view, _ ->
  // 设置当前View的指针视图为小手
  view.pointerIcon =
    PointerIcon.getSystemIcon(view.context,
      PointerIcon.TYPE_HAND)
  false
}
```

- **拖放**

在多窗口环境中，用户希望能够在应用之间拖放项目，比如直接把桌面的文件拖拽到应用中，或者把应用中的文件拖拽到桌面，亦或者是照片、音乐等等，如果需要添加拖放的话，可以去看下官方**拖放文档**(https://developer.android.google.cn/guide/topics/ui/drag-drop)中的说明进行操作，这里就不详细展开介绍了。

本文从屏幕适配到输入兼容性来说了下普通应用如何适配 Pad，有用的地方大家可以参考，当然如果能帮助到大家，哪怕是一点也足够了。就这样。

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

我为Android版Microsoft Edge所带来的变化

安卓13来了，快！扶起我来！

欢迎关注我的公众号

学习技术或投稿

▼

长按上图，识别图中二维码即可关注

阅读原文

喜欢此内容的人还喜欢

一个披着 Windows 外壳的轻量级 Linux 系统

21CTO

小诺 Snowy v2.0.2 已发布，更多是优化

小诺开源技术

来了，知名设计软件，2023全家桶!

果核剥壳