Android 适配全面解析，屏幕基础知识(尺寸、分辨率、dpi、dp/dip、sp)、适配方案(布局适配、控件适配、图片资源适配)、备用资源、机型适配、权限适配。

> 本文是我一点点归纳总结的干货，但是难免有疏忽和遗漏，希望不吝赐教。
> 转载请注明链接：https://blog.csdn.net/feather_wch/article/details/81484787

有帮助的话请点个赞！万分感谢！

# Android 适配

修正版本：2018/8/7-1(19：51)



# 屏幕适配

1、屏幕适配的是什么?

> 使得某一元素在Android不同尺寸、不同分辨率的手机上具备相同的显示效果

# 基础知识

## 屏幕尺寸

2、屏幕尺寸是什么?

> 1. 含义：手机对角线的物理尺寸
> 2. 单位：英寸（inch），1英寸=2.54cm
>    Android手机常见的尺寸有5寸、5.5寸、6寸等等

## 屏幕分辨率

3、屏幕分辨率是什么?

> 1. 含义：手机在横向、纵向上的像素点数总和(屏幕的"宽x高”=AxB)
> 2. 屏幕在横向方向（宽度）上有A个像素点，在纵向方向（高）有B个像素点
> 3. 实例：1080x1920，即宽度方向上有1080个像素点，在高度方向上有1920个像素点
> 4. 单位：px（pixel），1px=1像素点
> 5. UI设计师的设计图会以px作为统一的计量单位
> 6. Android手机常见的分辨率：320x480、480x800、720x1280、1080x1920、1080x2160

## 屏幕像素密度(dpi)

4、屏幕像素密度是什么?

> 1. 含义：每英寸的像素点数
> 2. 单位：dpi（dots per ich）
> 3. 假设设备内每英寸有160个像素，那么该设备的屏幕像素密度=160dpi

5、屏幕像素密度表

| 密度类型 | 代表的分辨率（px） | 屏幕像素密度（dpi） |
| --- | --- | --- |
| 低密度（ldpi） | 240x320 | 120 |
| 中密度（mdpi） | 320x480 | 160 |
| 高密度（hdpi） | 480x800 | 240 |
| 超高密度（xhdpi） | 720x1280 | 320 |
| 超超高密度（xxhdpi） | 1080x1920 | 480 |

6、屏幕尺寸、分辨率、像素密度三者关系

> 1. 像素密度 = 像素数 / 屏幕尺寸。这边涉及到勾股定理。
> 2. 公式： `像素密度=Sqrt(横向像素*横向像素+纵向像素*纵向像素)/屏幕尺寸` ， sqrt表示开方

## 密度无关像素(dp/dip)

7、 密度无关像素(dp/dip)

> 1. 含义：density-independent pixel，叫dp或dip，与终端上的实际物理像素点无关。
> 2. 单位：dp，可以保证在不同屏幕像素密度的设备上显示相同的效果
> 3. Android开发时用dp而不是px单位设置图片大小，是Android特有的单位
> 4. 在不同分辨率手机上，sp值会不同，但是dp值是相同的。

8、dp与px的转换

| 密度类型 | 代表的分辨率（px） | 屏幕像素密度（dpi） | 换算（px/dp） | 比例 |
| --- | --- | --- | --- | --- |
| 低密度（ldpi） | 240x320 | 120 | 1dp=0.75px | 0.75 |
| 中密度（mdpi） | 320x480 | 160 | 1dp=1px | 1 |
| 高密度（hdpi） | 480x800 | 240 | 1dp=1.5px | 1.5 |
| 超高密度（xhdpi） | 720x1280 | 320 | 1dp=2px | 2 |
| 超超高密度（xxhdpi） | 1080x1920 | 480 | 1dp=3px | 3 |

> 1. 因为ui设计师给你的设计图是以px为单位的，Android开发需要进行转换。

## 独立比例像素(sp)

9、独立比例像素(sp)

> 1. scale-independent pixel，叫sp或sip
> 2. 字体大小专用单位，会根据系统设置的字体大小进行缩放.
> 3. 字体放大会对APP的UI进行影响，因此建议用 `dp作为字体单位`

# 适配方案

10、 为什么要进行Android屏幕适配

> 1. Android系统碎片化：小米定制的MIUI、魅族定制的flyme、华为定制的EMUI等等
> 2. Android机型屏幕尺寸碎片化：5寸、5.5寸、6寸等等
> 3. Android屏幕分辨率碎片化：320x480、480x800、720x1280、1080x1920
> 4. `Android屏幕适配` 是为了保证某一元素在Android不同尺寸、不同分辨率的手机上具备相同的显示效果。

11、屏幕适配的本质?(2)

> 1. "布局"、"布局组件"需要匹配不同的 屏幕尺寸
> 2. "图片资源"匹配不同的 屏幕密度

12、基本适配技巧

> 1. 使用dp而非px
> 2. 少写固定尺寸(多用wrap_content, match_parent 与 weight 权重)
> 3. 使用相对布局，不要使用绝对布局
> 4. 自动拉伸的.9图
> 5. 使用shape代替纯色图片(相比于png、xml要更小)
> 6. 使用SVG矢量图替换位图

13、适配的方面

> 1. 布局适配: 最小宽度限定符、屏幕方向限定符
> 2. 控件适配：少用固定值宽高、尽量使用权重和百分比去安排控件。
> 3. 图片资源适配

## 备用资源

14、备用资源是什么?

> 1-Android提供 备用资源 这个概念主要适用于 适配 。
> 2-系统会根据设备的不同去自动加载对应文件夹里面的资源。比如：有布局目录为layout-sw600dp，如果是在宽度高于600dp的设备上就会去加载该目录的资源文件；如果宽度低于600dp就会去加载默认的layout目录中的布局。

| 手机特性 | 资源限定符 | 描述 |
| --- | --- | --- |
| 屏幕尺寸 | small | 小尺寸屏幕 |
| | normal | 正常大小的屏幕 |
| | large | 大尺寸屏幕 |
| | xlarge | 超大尺寸的屏幕 |
| 分辨率 | 320x240 | values-ldpi-320x240 |
| | 480x320 | values-480x320 |
| | 800x480 | values-800x480 |
| | 854x480 | values-854x480 |
| | 960x540 | values-960x540、values-hdpi-960x540 |
| | 960x640 | values-960x640、values-xhdpi-960x640 |
| | 1024x600 | values-mdpi-1024x600 |
| | 1280x720、1280x800 | values-1280x720 |
| | 其他 | values-xhdpi-1184x768、values-xhdpi-1280x720、values-xhdpi-1920x1080 |
| 密度 | ldpi | 低密度~120dpi |
| | mdpi | 中密度~160dpi |
| | hdpi | 高密度~240dpi、 |
| | xhdpi | 超高密度～320dpi 、 |
| | nodpi | 存放无视屏幕密度的资源，如：一些不能被拉伸的图片放在 drawable-nodpi ，但宽和高要写 wrap_content |
| | tvdpi | 主要用于电视，大多数App不需要用到 |
| 方向 | land | 横向屏幕 |
| | port | 纵向屏幕 |
| 版本 | v1~v19 | 新建工程时可以看见，API 1到API 19 |

备用资源-Google官方文档

15、备用资源在项目工程res目录中的体现

> drawable：默认Drawable目录
> drawable-mdpi：中密度-Drawable
> drawable-xxhdpi: 超高密度-Drawable
> drawable-en-hdpi: 英文-高密度-Drawable
> drawable-en-port：英文-纵向-Drawable
> drawable-port-notouch-12key：纵向-没有触摸屏-12键硬键盘-Drawable
> values：默认数值
> values-480x320: 480x320-values
> values-sw600dp: 最小宽度为600dp-values
> values-sw720dp-land：最小宽度720dp-横向-values
> values-v11：API为11-values
> values-v21：API为21-values
> values-xlarge：超大屏幕尺寸-vlues
> values-zh-rCN：中文-中国大陆-values

> - drawable、layout、values等大部分 `res/` 中的资源都可以用这些限定符进行控制。更多限定符要去看Google官方文档，且一定要按照官方的顺序去使用，如果顺序反了会导致无效。

16、使用脚本生成不同分辨率所需要的资源文件

> 1. Github地址：https://github.com/mengzhinan/PhoneScreenMatch
> 2. 因为UI提供的数值是按PX算的，所以在不同机型上 `px和dp的比例不同` ，可以使用脚本帮助我们一键生成。

## 布局适配

17、 布局的种类

> 1. 线性布局（Linearlayout）
> 2. 相对布局（RelativeLayout）
> 3. 帧布局（FrameLayout）
> 4. 绝对布局（AbsoluteLayout）
> 5. 约束布局（ConstraintLayout）

18、 布局优化方法

> 1. 禁用 `绝对布局（AbsoluteLayout）`
> 2. 尽可能使用约束布局和相对布局
> 3. 尽可能使用权重和百分比进行适配

19、使用约束布局（ConstraintLayout）

> 1. 约束布局提供了各种约束关系、比例关系很容易能解决适配问题。

**最小宽度限定符(layout-sw600dp)**

20、最小宽度限定符的作用

> 1. 通过指定某个最小宽度（ dp）来加载不同的UI资源
> 2. sw xxxdp，即small width的缩写，其不区分方向，即无论是宽度还是高度，只要大于xxxdp，就采用次此布局。(如：layout-sw600dp，无论是宽度还是高度，只要大于600dp，就采用layout-sw600dp目录下的布局)

21、最小宽度限定符的使用

> 1-手机的布局 `res/layout/main.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <fragment android:id="@+id/headlines"
            android:layout_height="fill_parent"
            android:name="com.example.android.newsreader.HeadlinesFragment"
            android:layout_width="match_parent" />
</LinearLayout>
```

> 2-平板的布局 `res/layout-sw600dp/main.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="400dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>
```

> 效果：
>
> 1. 最小宽度 ≥ 600 dp 的设备,系统会自动加载 layout-sw600dp/main.xml.
> 2. 最小宽度 < 600 dp 的设备,系统就会选择 layout/main.xml.

**屏幕方向（Orientation）限定符**

22、屏幕方向限定符作用

> 1. 根据布局方向 进行 布局的调整
>
> 适用场景：

| 屏幕 | 竖屏 | 横屏 |
| --- | --- | --- |
| 小屏幕 | 单面板 | 单面板 |
| 7 英寸平板电脑 | 单面板，带操作栏 | 双面板，宽，带操作栏 |
| 10 英寸平板电脑 | 双面板，窄，带操作栏 | 双面板，宽，带操作栏 |
| 电视 | | 双面板，宽，带操作栏 |

23、屏幕方向限定符的使用

> 1- res/layout/main.xml -手机小屏的 单面板

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent">

 <fragment android:id="@+id/headlines"
           android:layout_height="fill_parent"
           android:name="com.example.android.newsreader.HeadlinesFragment"
           android:layout_width="match_parent" />
</LinearLayout>
```

> 2- res/layout-sw600dp-port/main.xml - 7英寸,竖屏 的双面板(窄)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="200dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>
```

> 3- res/layout-sw600dp-land/main.xml - 7英寸,横屏 的双面板(宽)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="400dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>
```

## 控件适配

24、组件适配的方法

> 1. 使用 `match_parent` 、 `weight` 等进行控制。
> 2. 尽量通过 权重或者百分比 去安排控件的宽高。

## 图片资源适配

25、图片资源适配的作用

> 使得图片资源在不同屏幕密度上显示相同的像素效果

26、如何使用nine-patch图片进行适配

> 1. 使用 自动拉伸位图（`nine-patch`图片） ，后缀名是 `.9.png` ，它是一种被特殊处理过的PNG图片，设计时可以指定图片的拉伸区域和非拉伸区域；使用时，系统就会根据控件的大小自动地拉伸你想要拉伸的部分
> 2. 必须要使用 `.9.png` 后缀名，因为系统就是根据这个来区别nine-patch图片和普通的PNG图片
> 3. 当你需要在一个控件中使用nine-patch图片时,如 android:background="@drawable/button"系统就会根据控件的大小自动地拉伸你想要拉伸的部分

27、如何使用备用资源进行适配?

> 1. 根据常常遇到的手机机型的情况，去创建对应的资源目录，并且将美工提供的图片放置到对应目录中。
> 2. 具体可以参考 备用资源 在布局适配上面的使用。

# 18:9全面屏适配

28、声明最大屏幕高宽比

```
//AndroidManifest.xml
<application
    xxx>
    <meta-data android:name="android.max_aspect"
               android:value="ratio_float"/>
</application>
```

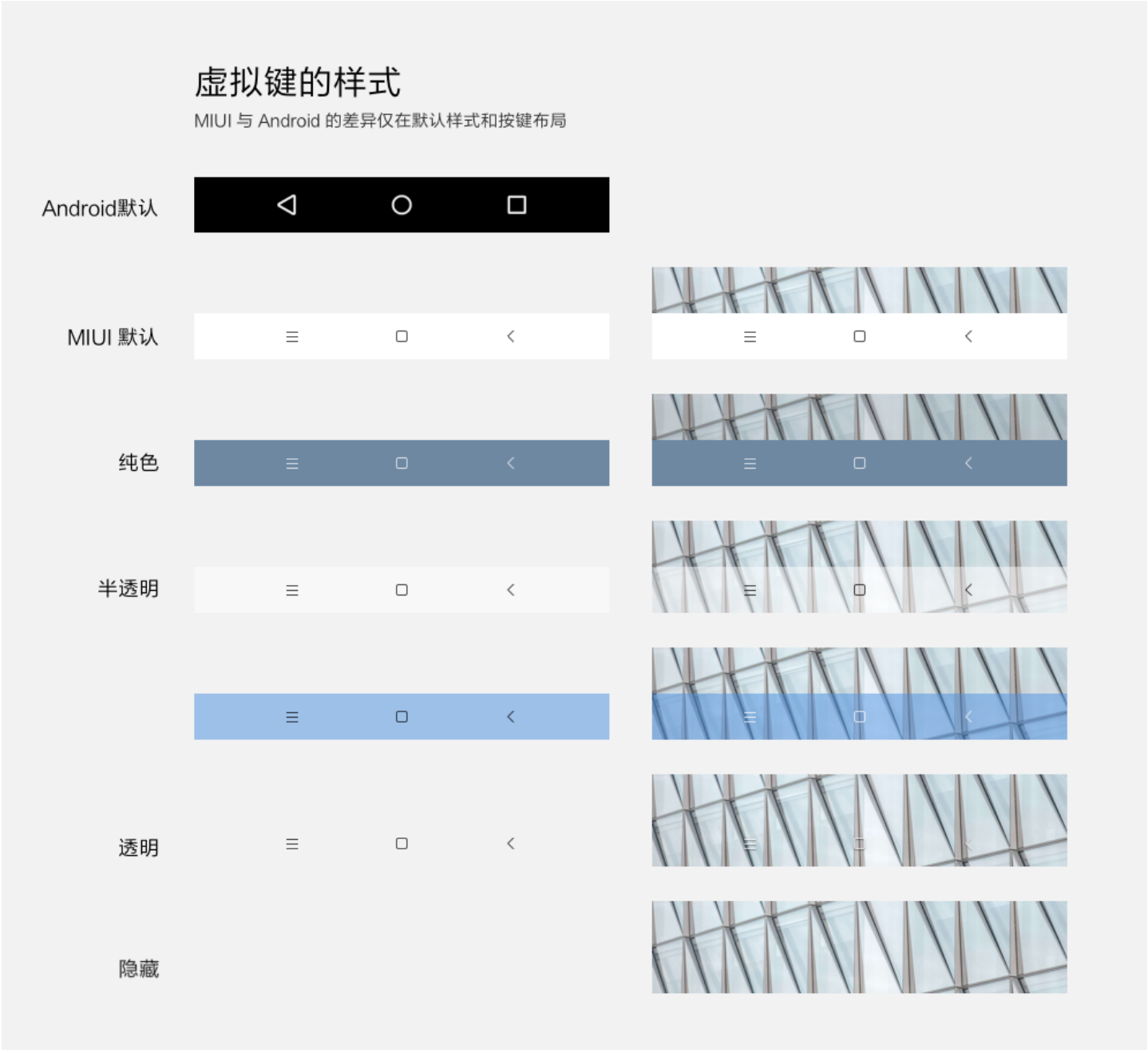
> 1. 在AndroidManifest.xml中添加 最大屏幕高宽比宽高比
> 2. 将其中 `ratio_float` 设置为需要的比例，可以使2.2或者更大，越大越好。
> 3. 这样能适配所有全面屏手机的 显示不全和黑边 问题。

29、给Application设置属性 `android:resizeableActivity` (表示是否支持多窗口显示)

```
<application
   android:resizeableActivity="true">
</application>
```

1. API24(Android 7.0)中添加的该属性。
2. 如果只通过该方法进行适配，如果因为默写原因 禁止了分屏模式 ， 会导致上下黑边。
3. 建议结合 最大屏幕宽高比 ， 适配度更高。

30、虚拟按键适配




1. 全面屏就必然有 虚拟按键 ， 但是虚拟按键的突兀感会破坏用户体验，最好的适配方案是选择合适的虚拟按键样式，保证视觉的统一性，达到一体化的沉浸体验。
2. 这些方法都只在Android 5.0以上生效，但是全面屏手机版本都在android 7.0以上。因此没有关系。

```
//方法一： window.setNavigationBarColor (int color)
public abstract void setNavigationBarColor(@ColorInt int color);
//方法二：在主体中添加设置项
<item name="android:navigationBarColor">要设置的颜色值</item>
```

### 31、启动页适配的问题

> 1. 统一采用 16:9 比例制作的引导页图片，放到 18:9 的屏幕中，显示效果会被拉伸。
> 2. 解决办法：
>    1. 新建资源目录： drawable-xxhdpi-2160x1080 或者 drawable-long 。在该资源目录下，放置对应的资源。
>    2. .9图 也能解决该问题。

## 刘海屏适配


### 32、华为刘海屏适配概述

> 1. 华为关于刘海屏适配提供了详细的官方文档：https://devcenter-test.huawei.com/consumer/cn/devservice/doc/50114
> 2. 华为默认处理逻辑(目的在于尽量减少开发者适配的工作量)：
>    1. 未设置meta-data值，页面横屏状态
>    2. 未设置meta-data值，页面竖屏状态，不显示状态栏
> 3. 结论如果APP没有进行适配，默认会偏移“刘海”的距离。

### 33、华为刘海屏适配步骤

> 1-配置meta-data

```
// 1.直接在AndroidManifest.xml的Application中添加meta-data.(对Application和Activity都生效---灰烬
<application
    xxx>
    <meta-data android:name="android.notch_support" android:value="true"/>
</application>

// 2. 在Activity中设置，则仅仅该Activity系统不会进行偏移处理（适配需要后续处理）
<activity
    xxx>
    <meta-data android:name="android.notch_support" android:value="true"/>
</activity>
```

> 2-检测是否存在刘海屏

```
public static boolean hasNotchInScreen(Context context) {
    boolean ret = false;
    try {
        ClassLoader cl = context.getClassLoader();
        Class HwNotchSizeUtil = cl.loadClass("com.huawei.android.util.HwNotchSizeUtil");
        Method get = HwNotchSizeUtil.getMethod("hasNotchInScreen");
        ret = (boolean) get.invoke(HwNotchSizeUtil);
    } catch (ClassNotFoundException e) {
        Log.e("test", "hasNotchInScreen ClassNotFoundException");
    } catch (NoSuchMethodException e) {
        Log.e("test", "hasNotchInScreen NoSuchMethodException");
    } catch (Exception e) {
        Log.e("test", "hasNotchInScreen Exception");
    } finally {
        return ret;
    }
}
```

> 3-获取刘海屏的参数(宽度+高度)

```
public static int[] getNotchSize(Context context) {
    int[] ret = new int[]{0, 0};
    try {
        ClassLoader cl = context.getClassLoader();
        Class HwNotchSizeUtil = cl.loadClass("com.huawei.android.util.HwNotchSizeUtil");
        Method get = HwNotchSizeUtil.getMethod("getNotchSize");
        ret = (int[]) get.invoke(HwNotchSizeUtil);
    } catch (ClassNotFoundException e) {
        Log.e("test", "getNotchSize ClassNotFoundException");
    } catch (NoSuchMethodException e) {
        Log.e("test", "getNotchSize NoSuchMethodException");
    } catch (Exception e) {
        Log.e("test", "getNotchSize Exception");
    } finally {
        return ret;
    }
}
```

> 4-UI适配：经过判断后，就可以进行UI适配。但是要保证重要内容显示在 安全区域 ； 可以将不重要内容或者被遮挡无所谓的内容放置到 危险区域

## 34、vivo&oppo适配

> 1-oppo官方文档: https://open.oppomobile.com/service/message/detail?id=61876
> 2-vivo官方文档: https://dev.vivo.com.cn/doc/document/info?id=103
> 3-仅仅有适配指导：如果有是具有刘海屏的手机，竖屏请显示状态栏，横屏不要在危险区显示重要信息或者设置点击事件。
> 4-Vivo判断是否是刘海屏手机：

```
public static final int NOTCH_IN_SCREEN_VOIO=0x00000020;//是否有凹槽
public static final int ROUNDED_IN_SCREEN_VOIO=0x00000008;//是否有圆角
public static boolean hasNotchInScreenAtVoio(Context context){
    boolean ret = false;
    try {
        ClassLoader cl = context.getClassLoader();
        Class FtFeature = cl.loadClass("com.util.FtFeature");
        Method get = FtFeature.getMethod("isFeatureSupport",int.class);
        ret = (boolean) get.invoke(FtFeature,NOTCH_IN_SCREEN_VOIO);

    } catch (ClassNotFoundException e) {
        Log.e("test", "hasNotchInScreen ClassNotFoundException");
    } catch (NoSuchMethodException e) {
        Log.e("test", "hasNotchInScreen NoSuchMethodException");
    } catch (Exception e) {
        Log.e("test", "hasNotchInScreen Exception");
    } finally {
        return ret;
    }
}
```

> 5-OPPO判断是否是刘海屏手机：

```
public static boolean hasNotchInOppo(Context context){
   return context.getPackageManager().hasSystemFeature("com.oppo.feature.screen.heteromorphism'
}
```

35、小米刘海屏适配

> 小米官方文档：https://dev.mi.com/console/doc/detail?pId=1160

# 机型适配

36、libxxx.so - text relocations问题

> 1. 问题现象： `targetSdkVersion >= 23` 会出现文法加载 `so文件` 并且报错。
> 2. 原因：Android 6.0(API23)开始，系统会禁止加载包含 `text relocations` 的共享库。
> 3. 解决办法1：有so源代码，根据Google官方指导手册可以进行解决。可以在使用NDK编译so的时候配置 `Android.mk` 添加配置项 `LOCAL_LDFLAGS += -fPIC` ，就能解决该问题。
> 4. 解决办法2：没有so源代码，那就不要使用该第三方so，及早更换没有问题的第三方库。

37、libxxx.so - W + E load segments are not allowed

> 1. 问题现象：在Android 8.0及其以上手机，出现加载 `so共享库` 报错。
> 2. 原因：从Android 8.0(API 27)开始，不允许有 `segment(段)` 的权限同时为 `W+E,可写，可执行` 。
> 3. 解决办法：避免这种情况，像 `R + E, R + W` 都是可以的。

> - Segment-段：在操作系统的内存管理层面，涉及到分段机制、分页机制。

38、在Service中启动StartActivity问题

> 1. 在荣耀Play中可以直接启动没有问题。
> 2. 在红米note等大部分机型上面，会报错，需要设置Flag为 `FLAG_ACTIVITY_NEW_TASK`
> 3. 在其他极少数机型上，出现了最近任务列表有两个相同app的情况。这时候就需要在AndroidManifest中给该Activity设置属性。

# 权限适配

39、Android 6.0以下的运行时权限验证问题

> 1. 在小米的一些机型上面，即使并不是Android 6.0以上版本，却依然需要运行时权限验证。
> 2. 是因为部分国产Room(MIUI 4.4以上)有自己的一套权限检查机制。
> 3. 解决办法：不支持我们所熟知的Android 6.0的那种动态权限申请。

40、Android 6.0/7.0的动态权限申请

> 1-权限检查

```
ContextCompat.checkSelfPermission();
```

> 2-权限申请

```
ActivityCompat.requestPermissions();// 多个权限可以一起申请
```

> 3-权限申请回调接口

```
onRequestPermissionsResult();
```

> 实现方法：Android 6.0 动态权限申请

41、Android 8.0的权限适配

> 1. Android 6.0: 同一权限组中任何一个权限被授权了，其他权限也自动被授权。如：READ_EXTERNAL_STORAGE和WRITE_EXTERNAL_STORAGE。
> 2. Android 8.0: 同一权限组中一个权限被授权后，同组其他权限不会再默认授予，需要进行申请。当然这次申请会直接通过。如果不在代码中显式申请，会直接崩溃。
> 3. 解决办法：权限分组，按组进行申请，以防止遗漏某一权限导致崩溃。

# 参考资料

1. 小猪屏幕适配
2. Android屏幕适配解决方案集合
3. Android手机 全面屏（18：9屏幕）适配指南
4. Android备用资源-Google文档
5. Android手机和平板Fragment适配
6. Android权限适配全攻略
7. Android 8.0权限适配
8. 详解刘海屏适配