Android屏幕适配全攻略

屏幕适配

目前市场上的主流分辨率

1280*720，800*480，854*480，1920*1080，960*540，480*320

一些重要概念

什么是屏幕尺寸、屏幕分辨率、屏幕像素密度？

- **屏幕尺寸**：屏幕尺寸指屏幕对角线的长度，单位是英寸，1英寸=2.54cm
- **屏幕分辨率**：屏幕分辨率值在横纵向上的像素点数，单位为px，1px=1个像素点，一般以纵向*横向像素，如1920*1080
- **屏幕像素密度**：像素密度是指每英寸上的像素点数，单位dpi(dot per inch)，屏幕像素密度与屏幕尺寸和屏幕分辨率有关

> 例如：Nexus 5 屏幕4.95inch 分辨率1920*1080
> 如何算它的像素密度呢？
> ()sqrt(1920^2+1080^2))/4.95=445DPI

什么是dp、dip、dpi、sp、px?之间的关系是什么？

- **px**:构成图像的最小单位，在开发中获取屏幕的宽高得到的就是px
- **dp、dip**:Density Independent Pixels的缩写，即密度无关像素，以160dpi为基准，1dip=1px
- **sp**:Scale-Independent Pixels 可以根据文字大小首选项进行缩放(12sp，14sp，18sp，22sp)
  什么是mdpi、hdpi、xdpi、xxdpi?如何计算和区分？
- **drawable-xhdpi**:不同分辨率的图片

名称	像素密度范围
mdpi	120dpi~160dpi
hdpi	160dpi~240dpi
xhdpi	240dpi~320dpi
xxhdpi	320dpi~480dpi
xxxhdpi	480dpi~640dpi

占比1：1.5：2：3：4

- **values-dimen**:同名对应的不同尺寸

解决方案

支持各种屏幕尺寸

- **使用wrap_content、match_parent、weight**
  weight的使用：要结合在某个方向上的长或宽为0dp(此时为权值越大越大)，要是为match_parent(此时权值越大越小)
- **使用相对布局，禁止用绝对布局**
  相对布局更加灵活
- **使用限定符**
  1.使用屏幕大小界定符

```
<?xml version="1.0" encoding="utf-8"?>
<!-- layout/main.xml  单面板 手机-->
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <fragment
        android:id="@+id/news"
        android:name="com.moyu.headfragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</LinearLayout>

<!-- layout-large/main.xml 双面板 如平板-->
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <fragment
        android:id="@+id/news"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:name="com.moyu.HeadFragment"/>
    <fragment
        android:id="@+id/detial"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:name="com.moyu.ContentFragment"/>
</LinearLayout>
```

> 注意：layout-large在3.2之前是这么用，在3.2后谷歌推出最小宽度界定符，如layout_sw600dp/main.xml

```
<!-- 优化管理布局-->
<!-- res/layout/main.xml 手机-->
<!-- res/layout-large/main.xml 2.3以前平板-->
<!-- res/layout-sw600/main.xml 2.3以后平板-->

<!-- res/layout/main.xml 单面板布局-->
<!-- res/layout/main_twopanes.xml 双面板布局-->

<?xml version="1.0" encoding="utf-8"?>

<!-- 默认布局 -->
<!-- res/values/layout.xml -->
<resources>
    <item name="main" type="layout">@layout/main</item>
    <!--设置has_two_panes这个boolean值主要是为了能在代码中有效控制在手机和
平板上的显示逻辑 -->
    <bool name="has_two_panes">false</item>
</resources>

<!-- Android3.2之前的平板布局-->
<!-- res/values-large/layout.xml -->
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
    <bool name="has_two_panes">true</item>
</resources>

<!-- Android3.2之后的平板布局 -->
<!-- res/values-sw600dp/layout.xml -->
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
    <bool name="has_two_panes">true</item>
</resources>
```

2.使用屏幕方向限定符

res/values-sw600dp-land/layouts.xml（横向水平）

res/values-sw600dp-port/layouts.xml（纵向）

```
<!-- 横向布局 -->
<!-- res/values-sw600dp-land/layout.xml -->
<resources>
    <item name="main" type="layout">@layout/main</item>
</resources>


<!-- 纵向布局-->
<!-- res/values-sw600dp=port/layout.xml -->
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

- **使用自动拉伸位图**
  .9图片，在图片周围添加一个像素的边界然后画一些黑色像素点；左上(相交部分)控制拉升区域，右下设置间隔区域(Padding box)[要使内容居中则需设置android：padding=“0dp”]即内容区域;点黑点处为拉升区域

支持各种屏幕密度

- **使用非密度制约像素**
  1.dp设置view大小
  2.sp设置字体大小
  3.更彻底的解决方案：
  因为分辨率不一样，所以不能用px；因为屏幕宽度不一样，所以要小心的用dp，那么我们可不可以用另外一种方法来统一单位，不管分辨率是多大，屏幕宽度用一个固定的值的单位来统计
  假设手机屏幕的宽度都是320某单位，那么我们将一个屏幕宽度的总像素数平均分成320份，每一份对应具体的像素就可以了
  生成资源value文件夹的代码

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;

public class MakeXml {
    //资源生成目录
    private final static String rootPath =
"C:\\Users\\Administrator\\Desktop\\layoutroot\\values-{0}x{1}\\";

    private final static float dw = 320f;
    private final static float dh = 480f;

    private final static String WTemplate = "<dimen name=\"x{0}\">
{1}px</dimen>\n";
    private final static String HTemplate = "<dimen name=\"y{0}\">
{1}px</dimen>\n";

    public static void main(String[] args) {
        makeString(320, 480);
        makeString(480,800);
        makeString(480, 854);
        makeString(540, 960);
        makeString(600, 1024);
        makeString(720, 1184);
        makeString(720, 1196);
        makeString(720, 1280);
        makeString(768, 1024);
        makeString(800, 1280);
        makeString(1080, 1812);
        makeString(1080, 1920);
        makeString(1440, 2560);
    }

    public static void makeString(int w, int h) {

        StringBuffer sb = new StringBuffer();
        sb.append("<?xml version=\"1.0\" encoding=\"utf-8\"?>\n");
        sb.append("<resources>");
        float cellw = w / dw;
        for (int i = 1; i < 320; i++) {
            sb.append(WTemplate.replace("{0}", i + "").replace("
{1}",
                    change(cellw * i) + ""));
        }
        sb.append(WTemplate.replace("{0}", "320").replace("{1}", w
+ ""));
        sb.append("</resources>");
```

```
        StringBuffer sb2 = new StringBuffer();
        sb2.append("<?xml version=\"1.0\" encoding=\"utf-8\"?
>\n");
        sb2.append("<resources>");
        float cellh = h / dh;
        for (int i = 1; i < 480; i++) {
            sb2.append(HTemplate.replace("{0}", i + "").replace("
{1}",
                    change(cellh * i) + ""));
        }
        sb2.append(HTemplate.replace("{0}", "480").replace("{1}",
h + ""));
        sb2.append("</resources>");

        String path = rootPath.replace("{0}", h + "").replace("
{1}", w + "");
        File rootFile = new File(path);
        if (!rootFile.exists()) {
            rootFile.mkdirs();
        }
        File layxFile = new File(path + "lay_x.xml");
        File layyFile = new File(path + "lay_y.xml");
        try {
            PrintWriter pw = new PrintWriter(new
FileOutputStream(layxFile));
            pw.print(sb.toString());
            pw.close();
            pw = new PrintWriter(new FileOutputStream(layyFile));
            pw.print(sb2.toString());
            pw.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
    public static float change(float a) {
        int temp = (int) (a * 100);
        return temp / 100f;
    }
}
```

我们将一个屏幕宽度分为320份，高度480份，然后按照实际像素对每一个单位进行复制，放在对应values-widthxheight文件夹下面的lax.xml和lay.xml里面，这样就可以统一所有你想要的分辨率的单位了

```
<?xml version="1.0" encoding="utf-8"?>
<resources><dimen name="x1">1.0px</dimen>
<dimen name="x2">2.0px</dimen>
<dimen name="x3">3.0px</dimen>
<dimen name="x4">4.0px</dimen>
<dimen name="x5">5.0px</dimen>
<dimen name="x6">6.0px</dimen>
<dimen name="x7">7.0px</dimen>
<dimen name="x8">8.0px</dimen>
<dimen name="x9">9.0px</dimen>
<dimen name="x10">10.0px</dimen>
...省略好多行
<dimen name="x300">300.0px</dimen>
<dimen name="x301">301.0px</dimen>
<dimen name="x302">302.0px</dimen>
<dimen name="x303">303.0px</dimen>
<dimen name="x304">304.0px</dimen>
<dimen name="x305">305.0px</dimen>
<dimen name="x306">306.0px</dimen>
<dimen name="x307">307.0px</dimen>
<dimen name="x308">308.0px</dimen>
<dimen name="x309">309.0px</dimen>
<dimen name="x310">310.0px</dimen>
<dimen name="x311">311.0px</dimen>
<dimen name="x312">312.0px</dimen>
<dimen name="x313">313.0px</dimen>
<dimen name="x314">314.0px</dimen>
<dimen name="x315">315.0px</dimen>
<dimen name="x316">316.0px</dimen>
<dimen name="x317">317.0px</dimen>
<dimen name="x318">318.0px</dimen>
<dimen name="x319">319.0px</dimen>
<dimen name="x320">320px</dimen>
</resources>
```

1080*1960分辨率下,由于1080和320是3.37倍的关系，所以x1=3.37px

```
<?xml version="1.0" encoding="utf-8"?>
<resources><dimen name="x1">3.37px</dimen>
<dimen name="x2">6.75px</dimen>
<dimen name="x3">10.12px</dimen>
<dimen name="x4">13.5px</dimen>
<dimen name="x5">16.87px</dimen>
<dimen name="x6">20.25px</dimen>
<dimen name="x7">23.62px</dimen>
<dimen name="x8">27.0px</dimen>
<dimen name="x9">30.37px</dimen>
<dimen name="x10">33.75px</dimen>
...省略好多行
</resources>
```

无论在什么分辨率下，x320都是代表屏幕宽度，y480都是代表屏幕高度。

具体使用

```
<!--在不同分辨率下都是显示这个样子-->
<!--定义一个120dp宽的按钮-->
<Button
    android:layout_width="@dimen/x120"
    android:layout_height="match_parent"/>
<!--定义一个160dp宽的按钮-->
<Button
    android:layout_width="@dimen/x180"
    android:layout_height="match_parent"/>
```

> 1.该解决方案需要对不同分辨率手机提供不同资源文件values-xxx*xxxx
> 2.这个解决方案的几个弊端，对于没有生成对应分辨率文件的手机，会使用默认values文件夹，如果默认文件夹没有，就会出现问题。

- **提供备用位图**
  由于 Android 可在具有各种屏幕密度的设备上运行，因此我们提供的位图资源应始终可以满足各类普遍密度范围的要求：低密度、中等密度、高密度以及超高密度。这将有助于我们的图片在所有屏幕密度上都能得到出色的质量和效果。
  要生成这些图片，我们应先提取矢量格式的原始资源，然后根据以下尺寸范围针对各密度生成相应的图片

> mdpi:1.0
> hdpi:1.5
> xhdpi:2.0
> xxhdpi:3.0
> xxxhdpi:4.0

从内存角度来看，要将对应分辨率的图片放到对应drawable的资源文件下，这样才不至于内存过大（引起原因：图片分辨率与资源文件夹不匹配时，在显示时就会对图片进行放缩，故耗内存）
因此在实际开发中尽量根据手机实际屏幕进行切图

实施自适应用户界面流着
前面我们介绍过，如何根据设备特点显示恰当的布局，但是这样做，会使得用户界面流程可能会有所不同。例如，如果应用处于双面板模式下，点击左侧面板上的项即可直接在右侧面板上显示相关内容；而如果该应用处于单面板模式下，点击相关的内容应该跳转到另外一个Activity进行后续的处理。所以我们应该按照下面的流程，一步步的完成自适应界面的实现。

- **确定当前布局**
  由于每种布局的实施都会稍有不同，因此我们需要先确定当前向用户显示的布局。例如，我们可以先了解用户所处的是“单面板”模式还是“双面板”模式。要做到这一点，可以通过查询指定视图是否存在以及是否已显示出来。

```
public class NewsReaderActivity extends FragmentActivity {
    boolean mIsDualPane;//判断单面板显示还是双面板显示

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_layout);

        View articleView = findViewById(R.id.article);
        mIsDualPane = articleView != null &&
                        articleView.getVisibility() ==
View.VISIBLE;
    }
}
```

- **根据当前布局做出响应**
  有些操作可能会因当前的具体布局而产生不同的结果。例如，在新闻阅读器示例中，如果用户界面处于双面板模式下，那么点击标题列表中的标题就会在右侧面板中打开相应报道；但如果用户界面处于单面板模式下，那么上述操作就会启动一个独立活动：

```
@Override
public void onHeadlineSelected(int index) {
    mArtIndex = index;
    if (mIsDualPane) {
        /* display article on the right pane */

mArticleFragment.displayArticle(mCurrentCat.getArticle(index));
    } else {
        /* start a separate activity */
        Intent intent = new Intent(this, ArticleActivity.class);
        intent.putExtra("catIndex", mCatIndex);
        intent.putExtra("artIndex", index);
        startActivity(intent);
    }
}
```

同样，如果该应用处于双面板模式下，就应设置带导航标签的操作栏；但如果该应用处于单面板模式下，就应使用下拉菜单设置导航栏。因此我们的代码还应确定哪种情况比较合适：

```
final String CATEGORIES[] = { "热门报道", "政治", "经济",
"Technology" };

public void onCreate(Bundle savedInstanceState) {
    ....
    if (mIsDualPane) {
        /* use tabs for navigation */

actionBar.setNavigationMode(android.app.ActionBar.NAVIGATION_MODE_
TABS);
        int i;
        for (i = 0; i < CATEGORIES.length; i++) {
            actionBar.addTab(actionBar.newTab().setText(
                CATEGORIES[i]).setTabListener(handler));
        }
        actionBar.setSelectedNavigationItem(selTab);
    }
    else {
        /* use list navigation (spinner) */

actionBar.setNavigationMode(android.app.ActionBar.NAVIGATION_MODE_
LIST);
        SpinnerAdapter adap = new ArrayAdapter(this,
                R.layout.headline_item, CATEGORIES);
        actionBar.setListNavigationCallbacks(adap, handler);
    }
}
```

- **重复使用其他活动中片段**
  多屏幕设计中的重复模式是指，对于某些屏幕配置，已实施界面的一部分会用作面板；但对于其他配置，这部分就会以独立活动的形式存在。例如，在新闻阅读器示例中，对于较大的屏幕，新闻报道文本会显示在右侧面板中；但对于较小的屏幕，这些文本就会以独立活动的形式存在。

在类似情况下，通常可以在多个活动中重复使用相同的 Fragment 子类以避免代码重复。例如，在双面板布局中使用了 ArticleFragment：

```
<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>
```

然后又在小屏幕的Activity布局中重复使用了它 ：

```
ArticleFragment frag = new ArticleFragment();
getSupportFragmentManager().beginTransaction().add(android.R.id.co
ntent, frag).commit();
```

- **处理屏幕配置变化**
  如果我们使用独立Activity实施界面的独立部分，那么请注意，我们可能需要对特定配置变化（例如屏幕方向的变化）做出响应，以便保持界面的一致性。
  也就是说，如果用户处于纵向模式下且屏幕上显示的是用于阅读报道的活动，那么就需要在检测到屏幕方向变化（变成横向模式）后执行相应操作，即停止上述活动并返回主活动，以便在双面板布局中显示相关内容：

```
public class ArticleActivity extends FragmentActivity {
    int mCatIndex, mArtIndex;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCatIndex = getIntent().getExtras().getInt("catIndex", 0);
        mArtIndex = getIntent().getExtras().getInt("artIndex", 0);
        // If should be in two-pane mode, finish to return to main
activity如果是双面板显示，就没有必要再显示内容Activity，因为一个Activity就
已经把两个面板包含了
        if (getResources().getBoolean(R.bool.has_two_panes)) {
            finish();
            return;
        }
        ...
}
```

最佳实践

关于高清设计图尺寸

Google官方给出的高清设计图尺寸有两种方案，一种是以mdpi设计，然后对应放大得到更高分辨率的图片，另外一种则是以高分辨率作为设计大小，然后按照倍数对应缩小到小分辨率的图片。
其实我们更推荐第二种方法，因为小分辨率在生成高分辨率图片的时候，会出现像素丢失，我不知道是不是有方法可以阻止这种情况发生。
而分辨率可以以1280*720或者是1960*1080作为主要分辨率进行设计。

ImageView的ScaleType属性

设置不同的ScaleType会得到不同的显示效果，一般情况下，设置为centerCrop能获得较好的适配效果。

动态设置

有一些情况下，我们需要动态的设置控件大小或者是位置，比如说popwindow的显示位置和偏移量等，这个时候我们可以动态的获取当前的屏幕属性，然后设置合适的数值

```
public class ScreenSizeUtil {
    //获取屏幕宽
    public static int getScreenWidth(Activity activity) {
        return
activity.getWindowManager().getDefaultDisplay().getWidth();
    }
    //获取屏幕高
    public static int getScreenHeight(Activity activity) {
        return
activity.getWindowManager().getDefaultDisplay().getHeight();
    }

}
```

总结

为什么要屏幕适配–>解决设备屏幕碎片化