移动应用软件开发

本次课内容：Android前端简易教程

王继良
软件学院

时间：周三（2）
地点：六教6A118

案例介绍

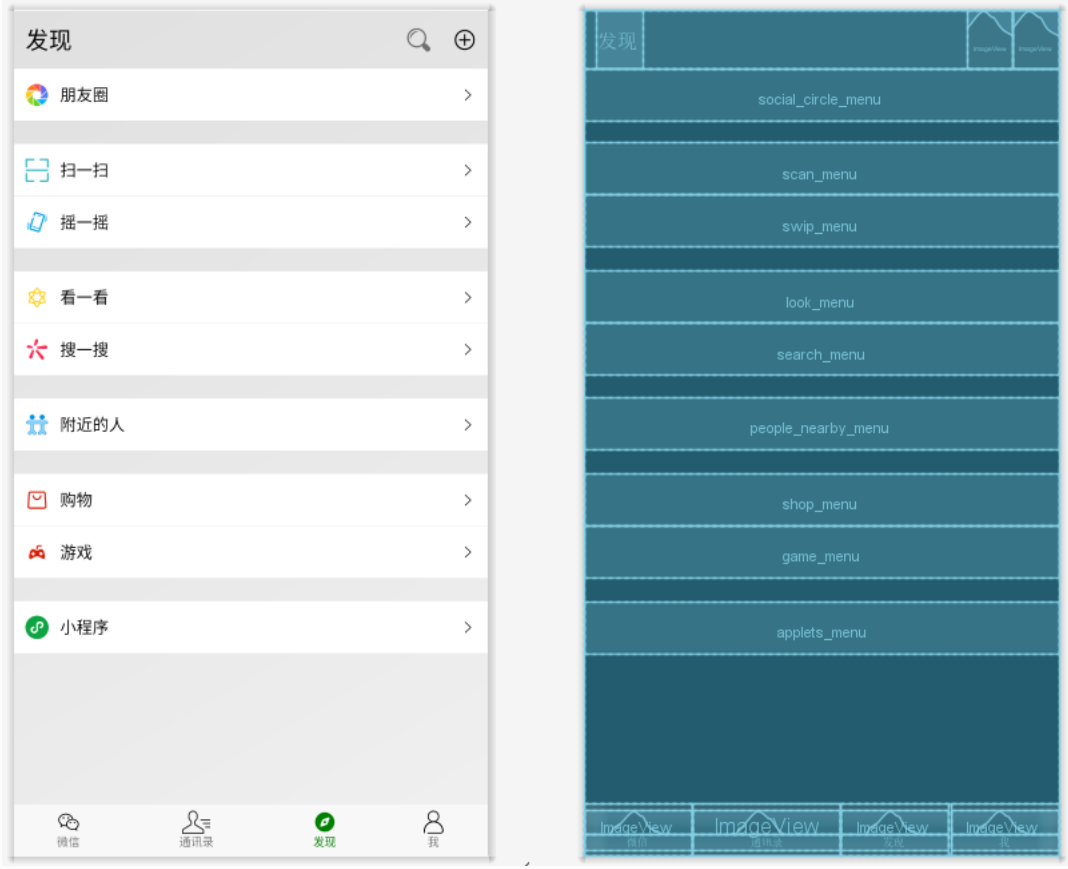
案例介绍：主界面

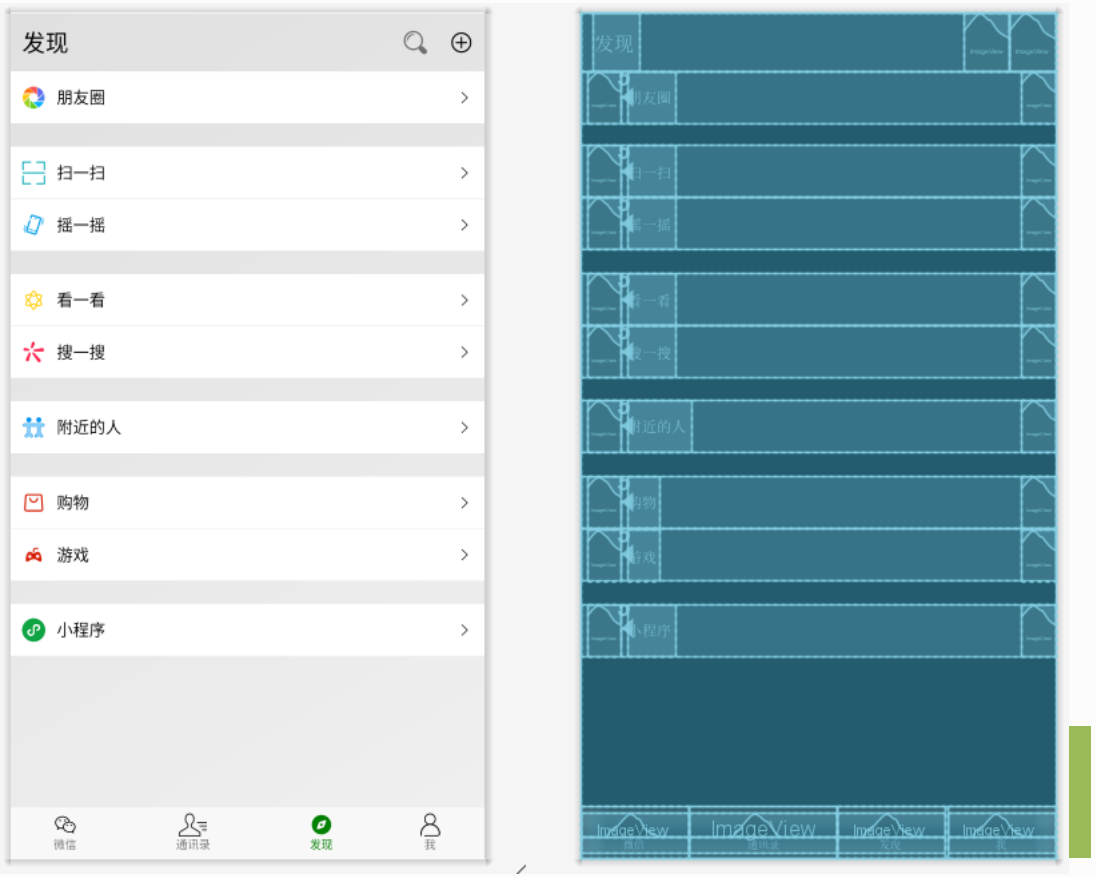
- 页面1，点击不同按钮后展现用不同组件构造方式构造的xml

- 页面2，点击按钮后展现不同方式加载的fragment

案例介绍：微信界面示例

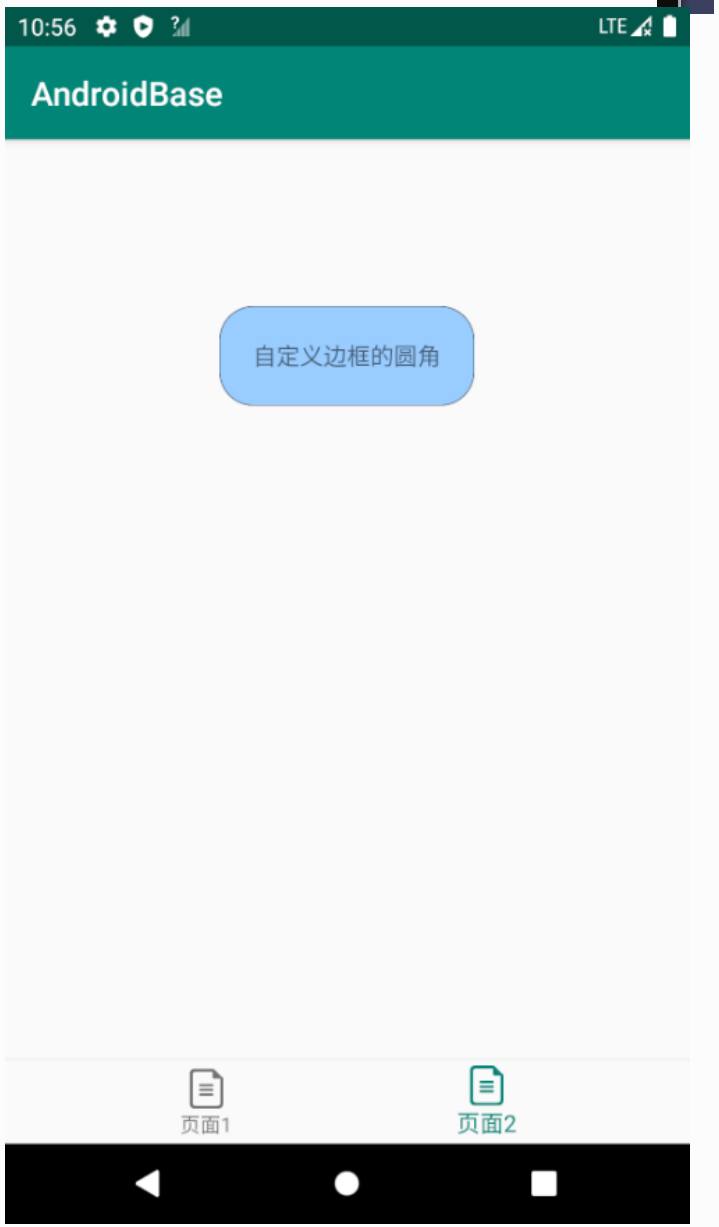
- 左图为封装view组件的activity和xml，右图则未封装



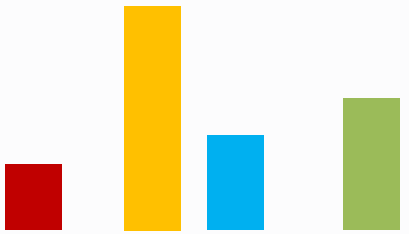


案例介绍：fragment

- 左图按钮点击后即可展示不同加载方式下的fragment
- 为展现更加直观，其中一种将按钮设置为不可见模式

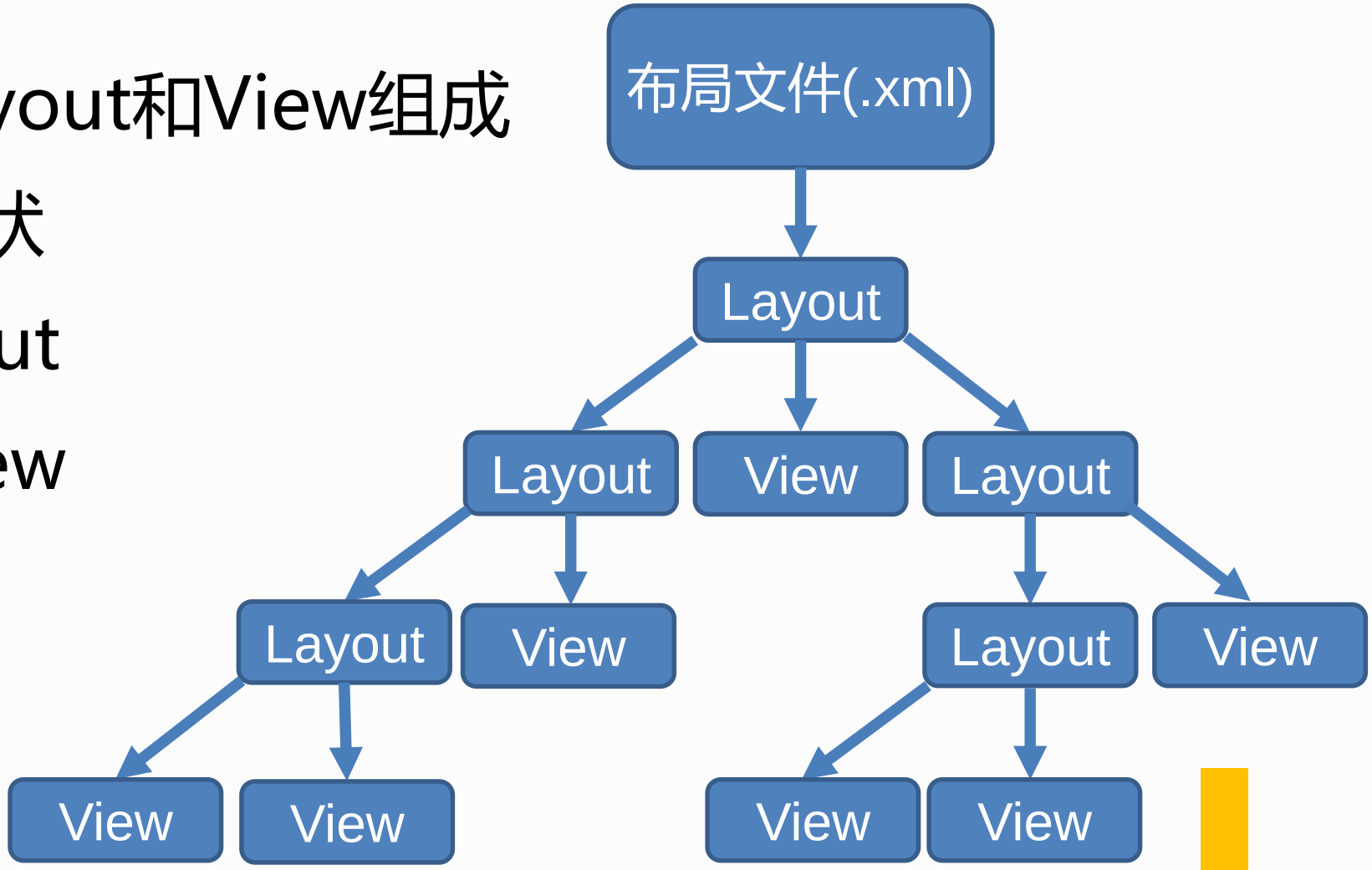


Layout

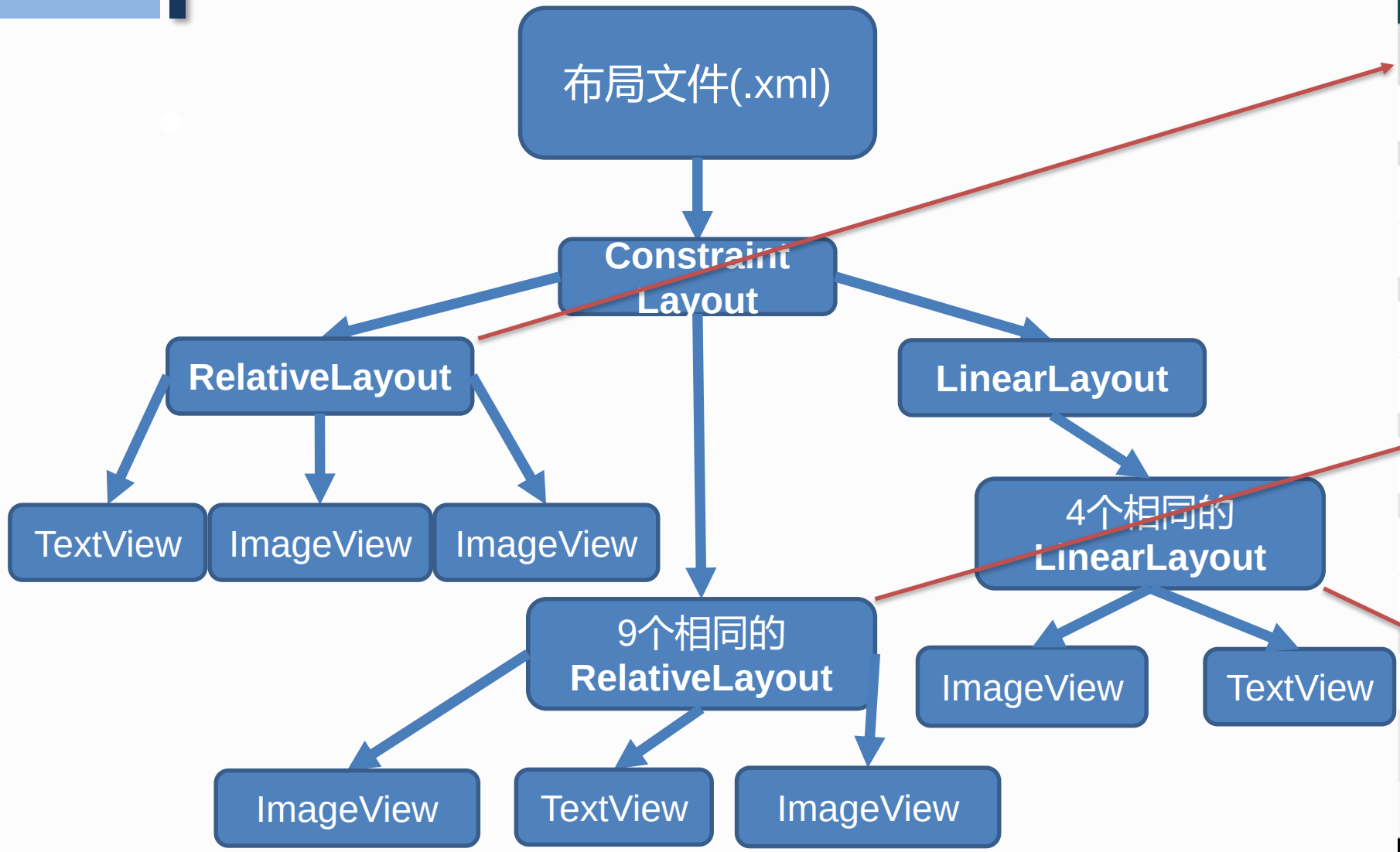


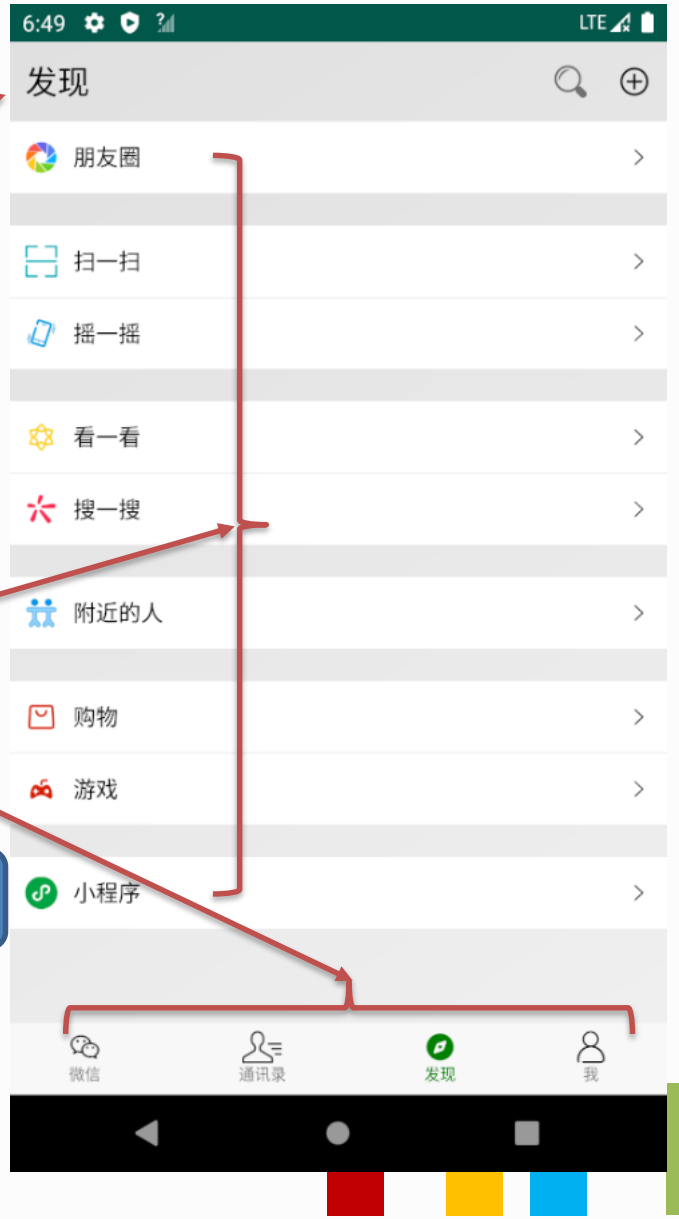
树形布局

- 布局文件由Layout和View组成
- 整体结构呈树状
- 根节点为Layout
- 叶子节点为View



布局介绍：以微信界面为例





Layout介绍: LinearLayout基础介绍

- **线性布局**是按照水平或垂直的顺序将子元素(可以是控件或布局)依次按照顺序排列，每一个元素都位于前面一个元素之后。
- 线性布局分为两种：水平方向和垂直方向的布局。分别通过属性android:orientation="vertical" 和 android:orientation="horizontal"来设置。
- android:layout_weight 表示子元素占据的空间大小的比例，这个值大小和占据空间反比。

Layout介绍: LinearLayout案例介绍

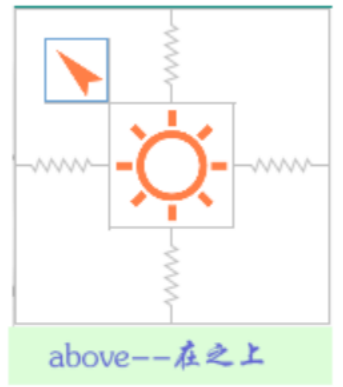
```
<LinearLayout
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="@dimen/bottom_menu_height">

    <LinearLayout
        android:id="@+id/wechat_menu"
        android:background="#f8f8f8"
        android:orientation="vertical"
        android:layout_weight="1"
        android:paddingTop="@dimen/bottom_menu_padding"
        android:layout_width="wrap_content"
        android:layout_height="@dimen/bottom_menu_height">
            <ImageView
                android:src="@drawable/ic_wechat"
                android:layout_width="wrap_content"
                android:layout_height="@dimen/
                        bottom_menu_icon_height" />
            <TextView
                android:text="微信"
                android:textSize="@dimen/bottom_menu_text_size"
                android:textAlignment="center"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" />
        </LinearLayout>
        <LinearLayout 通讯录… </LinearLayout>
        <LinearLayout 发现… </LinearLayout>
        <LinearLayout 我…</LinearLayout>
</LinearLayout>
```

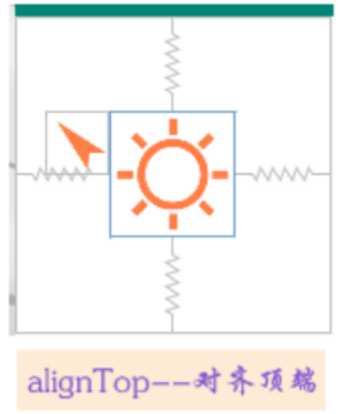
Layout介绍: RelativeLayout基础介绍

- **相对布局**是相互之间相关位置或者和他们的parent位置相关，参照控件可以是父控件，也可以是其他子控件，
- 被参照的控件必须要在参照它的控件之前定义，否则将出现异常。
- 相对布局模型所涉及的属性设置比较多，但并不复杂。最灵活，非常适合于一些比较复杂的界面设计。

Layout介绍: RelativeLayout常用位置属性

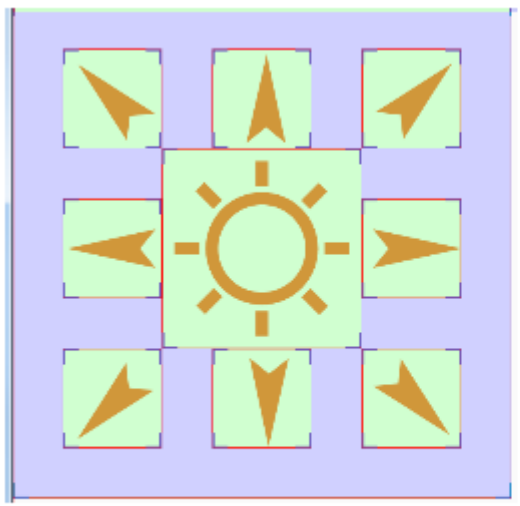
android:layout_toLeftOf	位于引用组件的左方
android:layout_toRightOf	位于引用组件的右方
android:layout_above	位于引用组件的上方
android:layout_below	位于引用组件的下方
android:layout_alignParentLeft	对齐父组件的左端
android:layout_alignParentRight	对齐父组件的右端
android:layout_alignParentTop	对齐父组件的顶部
android:layout_alignParentBottom	对齐父组件的底部
android:layout_centerInParent	相对于父组件居中
android:layout_centerHorizontal	横向居中
android:layout_centerVertical	垂直居中



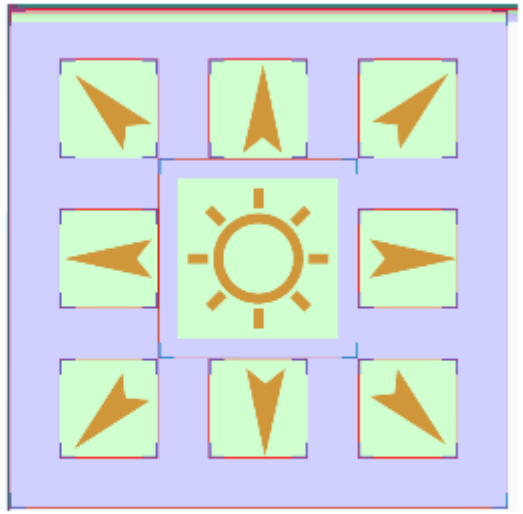


Layout介绍: RelativeLayout边缘属性

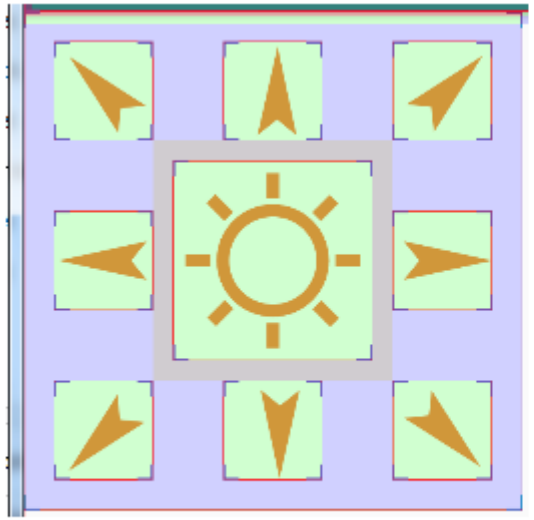
- padding和margin，两者都可以让自己与旁边的控件产生间隙，区别在于：padding缩小自己，margin推开其他控件。



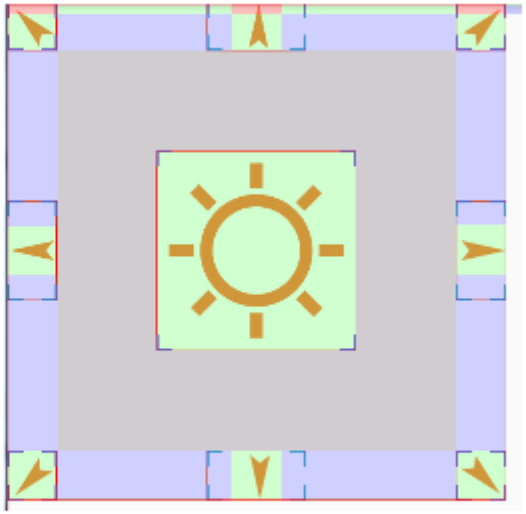
原图



padding=10dp



layout_margin=10dp

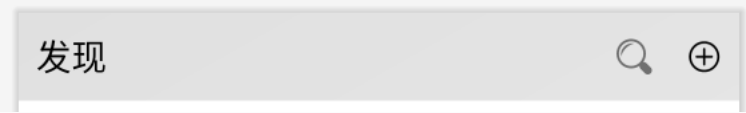


layout_margin=50dp

Layout介绍: RelativeLayout案例介绍

```
<RelativeLayout
        android:id="@+id/top_panel"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:textSize="20sp">

<TextView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:textColor="#000000"
            android:gravity="center_vertical"
            android:layout_marginStart="10dp"
            android:textSize="20sp"
            android:text="发现" />
```

```
<ImageView
            android:id="@+id/add_button"
            android:src="@drawable/ic_add"
            android:padding="10dp"
            android:scaleType="fitCenter"
            android:layout_alignParentEnd="true"
            android:layout_height="wrap_content"
            android:layout_width="40dp" />
 <ImageView
            android:id="@+id/search_button"
            android:layout_toStartOf="@id/add_button"
            android:src="@drawable/ic_search"
            android:padding="10dp"
            android:layout_width="40dp"
            android:layout_height="wrap_content" />
 </RelativeLayout>
```



Layout介绍: ConstraintLayout基础介绍

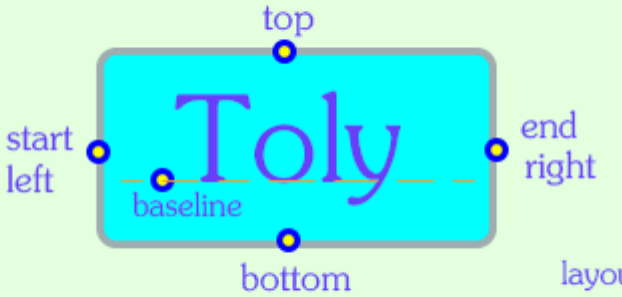
- **约束布局**在Api9开始出现，从 Android Studio 2.3 起，官方的模板默认使用。
- 它的出现主要是为了解决布局嵌套过多的问题，以灵活的方式定位和调整小部件。嵌套得越多，设备绘制视图所需的时间和计算功耗也就越多。
- 可以按照比例约束控件位置和尺寸，能够更好地适配屏幕大小不同的机型。

Layout介绍: ConstraintLayout定位属性

ConstraintLayout定位属性一览

layout_constraintLeft_toLeftOf
layout_constraintLeft_toRightOf
layout_constraintRight_toLeftOf
layout_constraintRight_toRightOf

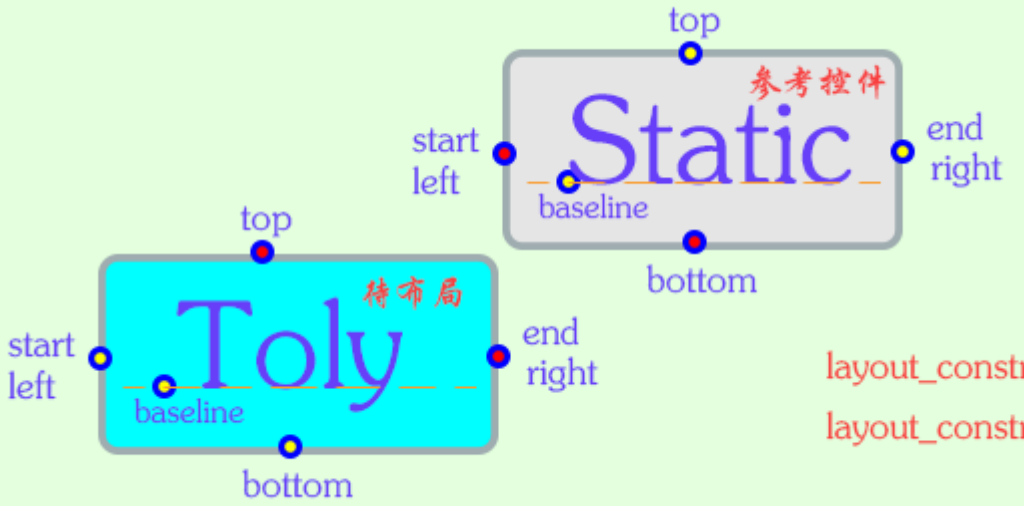
layout_constraintTop_toTopOf
layout_constraintTop_toBottomOf
layout_constraintBottom_toTopOf
layout_constraintBottom_toBottomOf



layout_constraintStart_toStartOf
layout_constraintStart_toEndOf
layout_constraintEnd_toStartOf
layout_constraintEnd_toEndOf

layout_constraintBaseline_toBaselineOf

待布局控件的XX到参考控件的XX

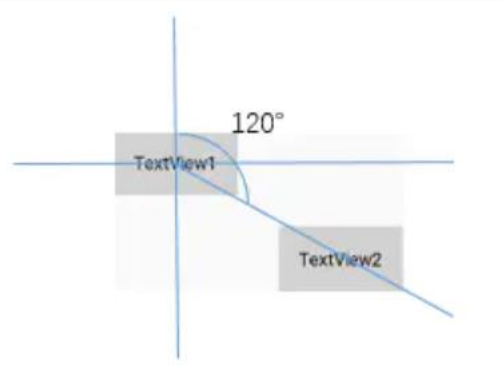


layout_constraintEnd_toStartOf
layout_constraintTop_toBottomOf

Layout介绍: ConstraintLayout定位属性

- **角度**

  app:layout_constraintCircle="@+id/TextView1"
  app:layout_constraintCircleAngle="120"（角度）
  app:layout_constraintCircleRadius="150dp"（距离）

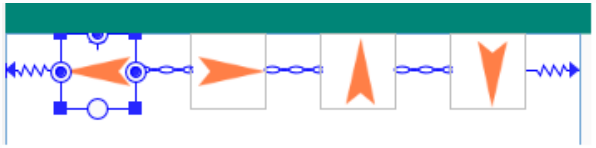


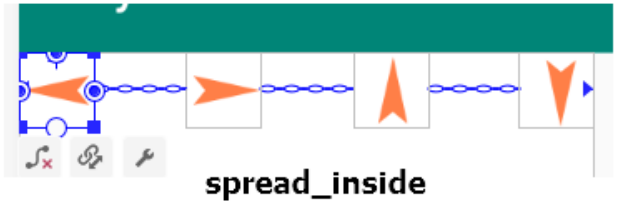
- **边距**margin同relativelayout
- **偏移**（固定值与比例值）

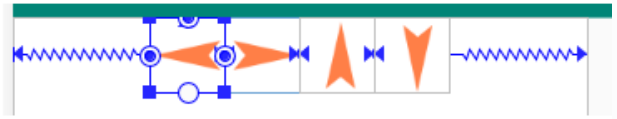
  android:layout_marginLeft="100dp“
  app:layout_constraintHorizontal_bias="0.3"

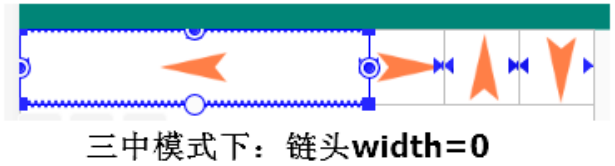
Layout介绍: ConstraintLayout定位属性

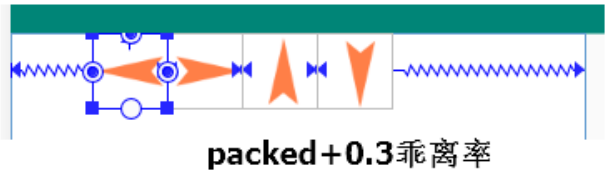
- **控件链：**加在链头，除首位节点，其他都持有前后的引用，一个接着一个，后面有连到前面的结构。



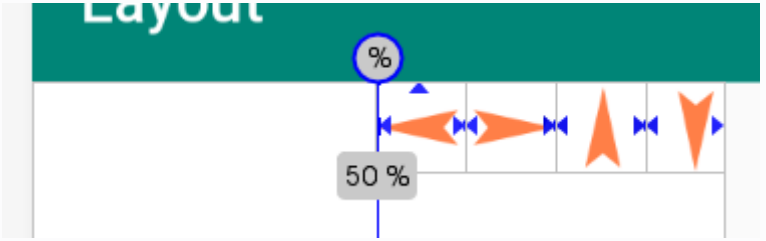


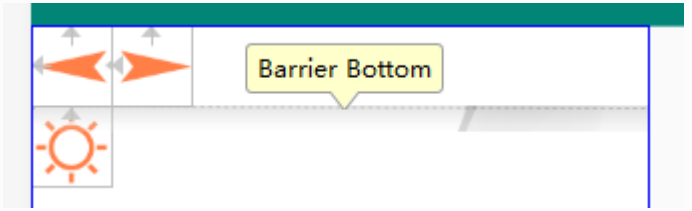






- **不可视的辅助标签：Guideline：**保留自己的位置信息，为布局提供参考，**Barrier：**可看作一个透明的墙，能提供约束。

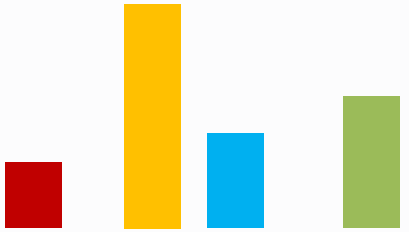




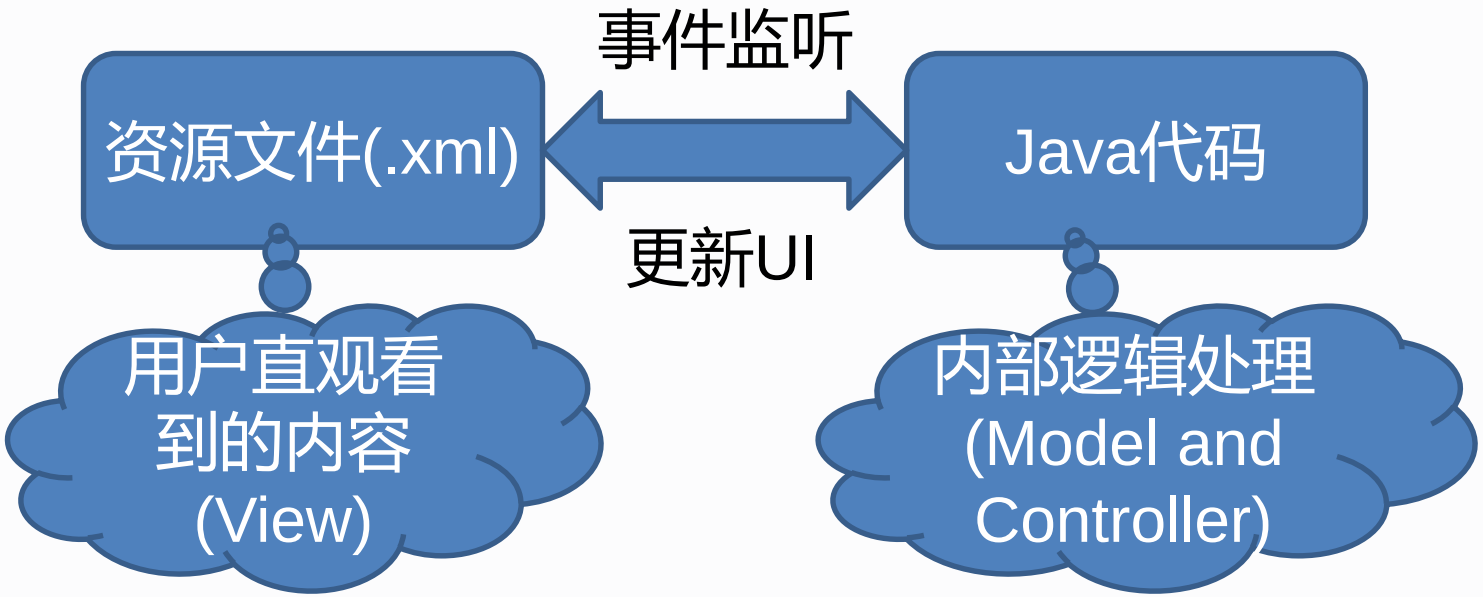
Layout介绍：不常用类型基础介绍

- FrameLayout：将所有的子元素放在整个界面的左上角，后面的子元素直接覆盖前面的子元素。
- TableLayout：表格布局，适用于多行多列的布局格式，类似于html的table；模型以行列的形式管理子控件，每一行为一个TableRow或View的对象。TableRow可添加子控件，每添加一个子控件为一列。
- AbsoluteLayout：绝对布局中将所有的子元素通过设置横纵坐标来固定位置，不能自适应屏幕尺寸大小，所以应用得相当少。

项目框架分析

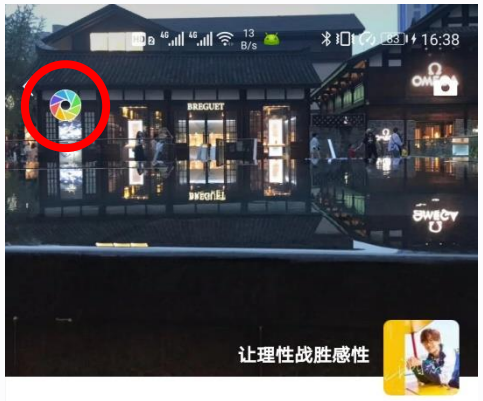


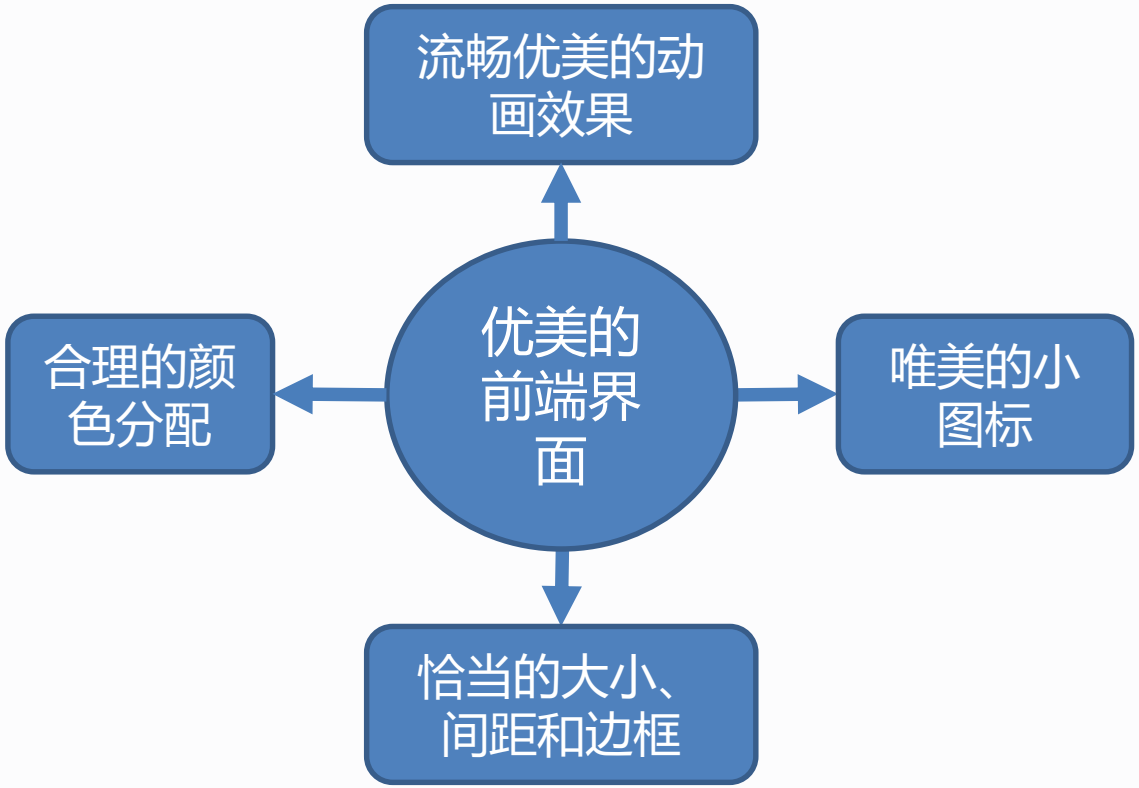
Android项目框架



优美的前端界面


发现
朋友圈
背景用白色
扫一扫
摇一摇
看一看
搜一搜
附近的人
购物
游戏
小程序
文字用黑色
间距用灰色
微信
通讯录
发现
我


让理性战胜感性


流畅优美的动画效果
合理的颜色分配
优美的前端界面
唯美的小图标
恰当的大小、间距和边框


邵阳
查看我的花呗额度
扫一扫
付钱
收钱
卡包
外卖
余额宝
酒店住宿
电影演出
市民中心
转账
信用卡还款
充值中心
芝麻信用
滴滴出行
校园生活
火车票机票
蚂蚁森林
健康码
更多
你有14条未读消息
4小时前
瓜分1亿体验金
余利宝限时免费发放
立即抢
抗击新冠肺炎
较上日新增确诊数据
0
湖南新增
21
国内新增
9
境外输入新增
89621
国外新增
海外动态
踏青好去处
交通出行
防护必买
好店榜单
首页
理财
口碑
朋友
我的

Manifest文件的改动

- 对所需要的权限提前进行申请

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

- 部分权限必须在运行时动态申请，否则默认为禁止状态！哪怕已经在AndroidManifest文件中声明，也不会弹出授权框。

```
if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED)
    ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, requestCode: 100);
```

- 对新添加的activity也要进行申明

```
<activity android:name=".FalseWechat.FalseWeChat" />
<activity android:name=".FalseWechat.FalseWeChat2" />
<activity android:name=".TemplateActivity1" />
```

res目录的结构与配置

anim:存放动画的描述文件

drawable:存放各类图形的描述文件，包括drawable的描述文件，以及三种图片格式：png（推荐）、jpg（支持）、gif（不推荐，因为ImageView只显示gif的第一帧）。

layout:存放页面的布局文件，主要在Acitivity、Fragment以及部分自定义控件中使用

menu:存放菜单的布局文件

raw:存放原始格式的文件，一般是二进制的流文件，比如音频文件、视频文件等等

mipmap:图片资源（google推荐只存启动图标）

values:存放各类参数的配置文件，具体的配置文件说明如下

——attrs.xml:存放自定义控件的属性信息

——colors.xml:存放颜色的定义文件

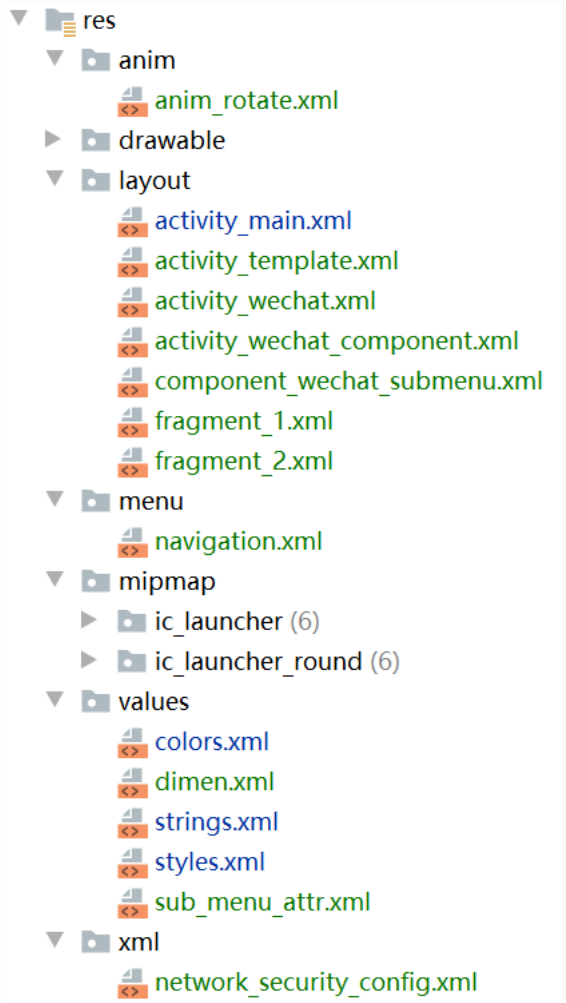
——dimens.xml:存放像素的定义文件

——ids.xml:存放控件id的定义文件

——strings.xml:存放字符串类型的定义文件

——styles.xml:存放控件风格的定义文件

xml:存放其他的xml文件



如何寻找素材

- 访问阿里巴巴矢量图标库或其他素材库，搜索你想要的图标名称，可以调整图标的大小和颜色，然后点击SVG下载

矢量图添加方法

- 右键点击res下的drawable文件夹，new->Vector Asset
- 在Path中选择刚才下载的svg文件，然后在Name中命名，然后导入
- 然后就可以在drawable中看到刚才导入的图片xml文件
- 在布局文件中通过ImageView设置src属性就可以展示出来

动态朋友圈图标实现方法

- 在anim_rotate.xml中定义

```
<rotate
android:drawable="@drawable/ic_social_circle"
        android:fromDegrees="0"
        android:toDegrees="359"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="1000"
        android:repeatCount="-1">
    </rotate>
```

- 在FalseWeChat中调用

```
Animation rotate = AnimationUtils.loadAnimation(this,R.anim.anim_rotate);
if (rotate != null) {
    social_circle.startAnimation(rotate);
}  else {
    social_circle.setAnimation(rotate);
    social_circle.startAnimation(rotate);
```

自定义圆角

- 在drawable中新建back_corner.xml

```
<shape
xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#99CCFF" />
    <corners android:topLeftRadius="20dp"
        android:topRightRadius="20dp"
        android:bottomRightRadius="20dp"
        android:bottomLeftRadius="20dp"/>
    <stroke android:width="0.5dp" android:color="#000000" />
</shape>
```

利用自定义圆角实现控件

```
<TextView
    android:layout_marginTop="100dp"
    android:background="@drawable/back_corner"
    android:layout_gravity="center_horizontal"
    android:padding="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="自定义边框的圆角" />
```

界面元素的位置和大小

高：45dp
宽：match_parent

- 界面元素的位置主要由外层的Layout影响，界面元素可以通过内部标签设置自己的位置，但是不同的Layout对界面元素的位置限制很大。
- 元素大小通过layout_width和layout_height设置，文字大小通过textSize设置
  - android:layout_height="45dp"
  - android:layout_width= “match_content"
  - android:textSize="20sp"

界面元素的背景边框，内外间距和颜色

- 通过背景(background)来设置边框
  - <padding android:top="0.5dp “ />

边框：0.5dp

- 通过背景设置渐变色，textcolor设置字体颜色
  - android:background="#f8f8f8"
  - android:textColor="#008000"

间距：20dp

- 通过margin和padding来设置外部和内部间距
  - android:layout_marginStart="10dp"
  - android:padding= “20dp"

Value的基本用法

- 为什么要通过专门的文件去存放各类参数的配置呢?
  - 如果在工程中定义button或textView样式，这些样式中包含着文字的大小，背景图片，前置图片等一些资源，而且会在很多地方用到它。
  - 如果把文字大小，图片样式等写在XML中或者代码中会非常不利于维护，一旦要修改的时候，需要挨个文件找，挨个修改。
  - 所以利用value去维护时，只需要修改对应的文件里定义的值。所有引用它的地方都会自动的修改这样，以达到资源重复利用的维护目的。

Value的实现代码

- 像素

```
<dimen name="bottom_menu_text_size">10sp</dim
<dimen name="sub_menu_height">45dp</dimen>
<dimen name="sub_menu_gap">20dp</dimen>
```

- 字符串

```
<string name="app_name">AndroidBase</string>
<string name="add_text">相加</string>
```

- 颜色

```
<color name="colorPrimary">#008577</color>
<color name="colorPrimaryDark">#00574B</col
<color name="colorAccent">#D81B60</color>
```

```
<RelativeLayout
    android:id="@+id/swip_menu"
    app:layout_constraintTop_toBottomOf="@id/scan_menu"
    android:background="@drawable/back_top_line"
    android:layout_width="match_parent"
    android:layout_height="@dimen/sub_menu_height">
```

```
<application
    android:networkSecurityConfig="@xml/network_sec
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
```

```
<item name="colorPrimary">@color/colorPrimary</item>
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
```

Menu的基本用法

- 用来定义菜单项，里面的属性主要有icon、showAsAction、title等。
- 菜单资源文件必须放在res/menu目录中。菜单资源文件必须使用<menu>标签作为根节点。除了<menu>标签外，还有另外两个标签<item>和<group>用于设置菜单项和分组
- <menu>标签没有任何属性，但可以嵌套在<item>标签中，表示子菜单的形式。不过<item>标签中不能再嵌入<item>标签。

Menu的实现代码

- 定义

```
<item android:id="@+id/navigation1"
      android:icon="@drawable/ic_page"
      android:title="页面1" />
```

- 使用BottomNavigationView将menu展示在页面下方

如右图所示

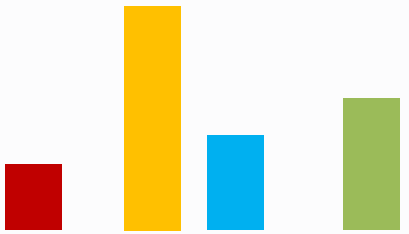
```
<com.google.android.material.bottomnavigation.
BottomNavigationView
    android:id="@+id/navigation"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="?android:attr/windowBackground"
    app:menu="@menu/navigation"
    tools:ignore="MissingConstraints" />
```

Menu的实现代码

- 调用

```
navigationMenu.setOnNavigationItemSelectedListener(item -> {
    FragmentTransaction trans = fm.beginTransaction();
    switch (item.getItemId()) {
        case R.id.navigation1:
            trans.show(fragment1).hide(fragment2).commit();
            return true;
        case R.id.navigation2:
            trans.show(fragment2).hide(fragment1).commit();
            return true;
    }
    return false;
});
```

Fragment



Fragment介绍

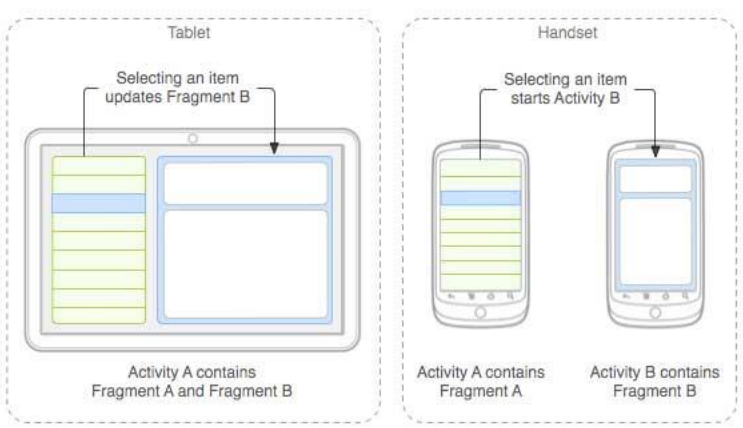
- Fragment 是activity的一部分，使得activity更加的模块化设计。可以认为fragment是一种子activity。
- 通过继承 Fragment 类来创建fragment。可以通过使用元素在activity的布局文件中声明fragment来在activity中插入fragment。
- 在引入fragment之前，由于每次给定的一个时间点在屏幕上只能显示单一的activity，因此无法分割设备屏幕并且独立的控制不同的部分。fragment的引入带来了更大的灵活性，并使得一个时间点只能在屏幕上有一个单一activity的限制被移除。现在可以有单一的activity，但每个activity由多个fragment组装，每个fragment有自己的布局，事件和完整的生命周期。

Fragment特性

- fragment拥有自己的布局，自己的行为及自己的生命周期回调。
- 当activity在运行的时候，可以在activity中添加或者移除fragment。
- 可以合并多个fragment在一个单一的activity中来构建多栏的UI。
- fragment可以被用在多个activity中。
- fragment的生命周期和它的宿主activity紧密关联。这意味着activity被暂停，所有activity中的fragment被停止。
- fragment可以实现行为而没有用户界面组件。
- fragment是 Android API 版本11中被加入到 Android API。

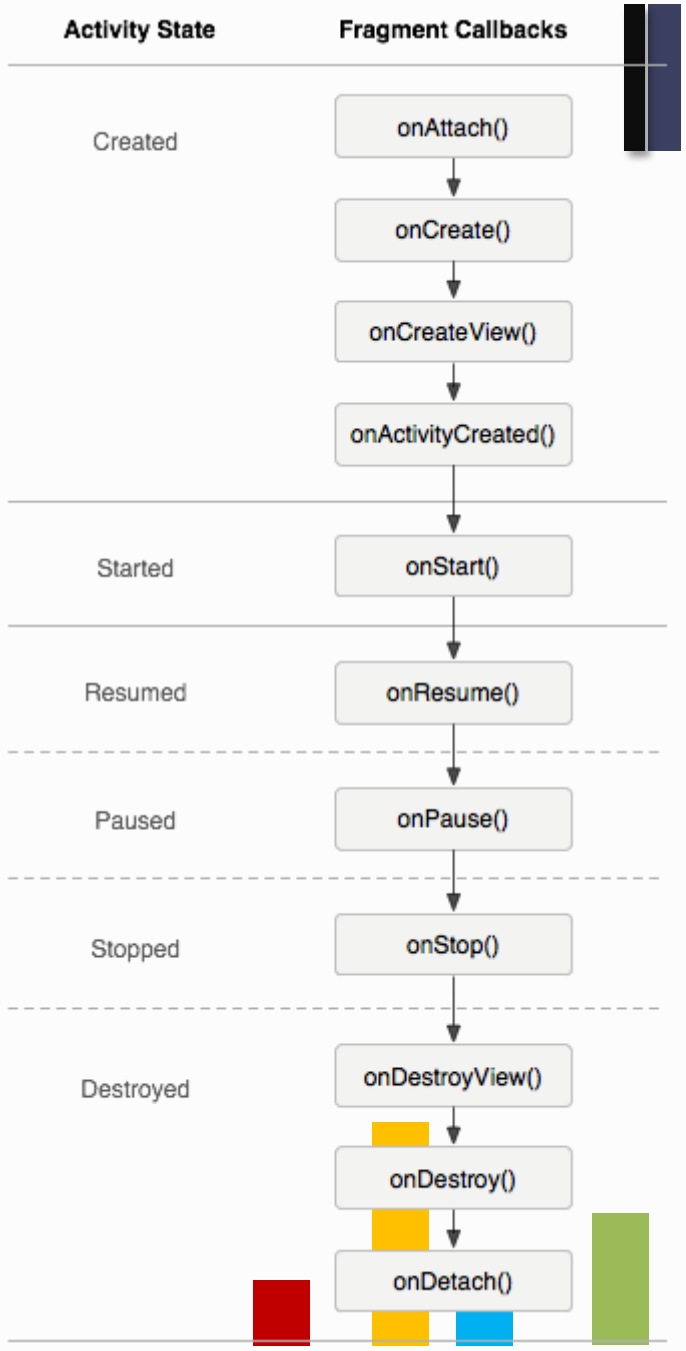
Fragment作用举例

- 当运行在在平板尺寸的设备上，这个应用程序可以在activity A中嵌入两个fragment。在手机设备屏幕上，由于没有足够的空间，activity A仅包含有文章列表的fragment，当用户点击文章时，启动包含第二个fragment的activity B来阅读文章。



Fragment的生命周期

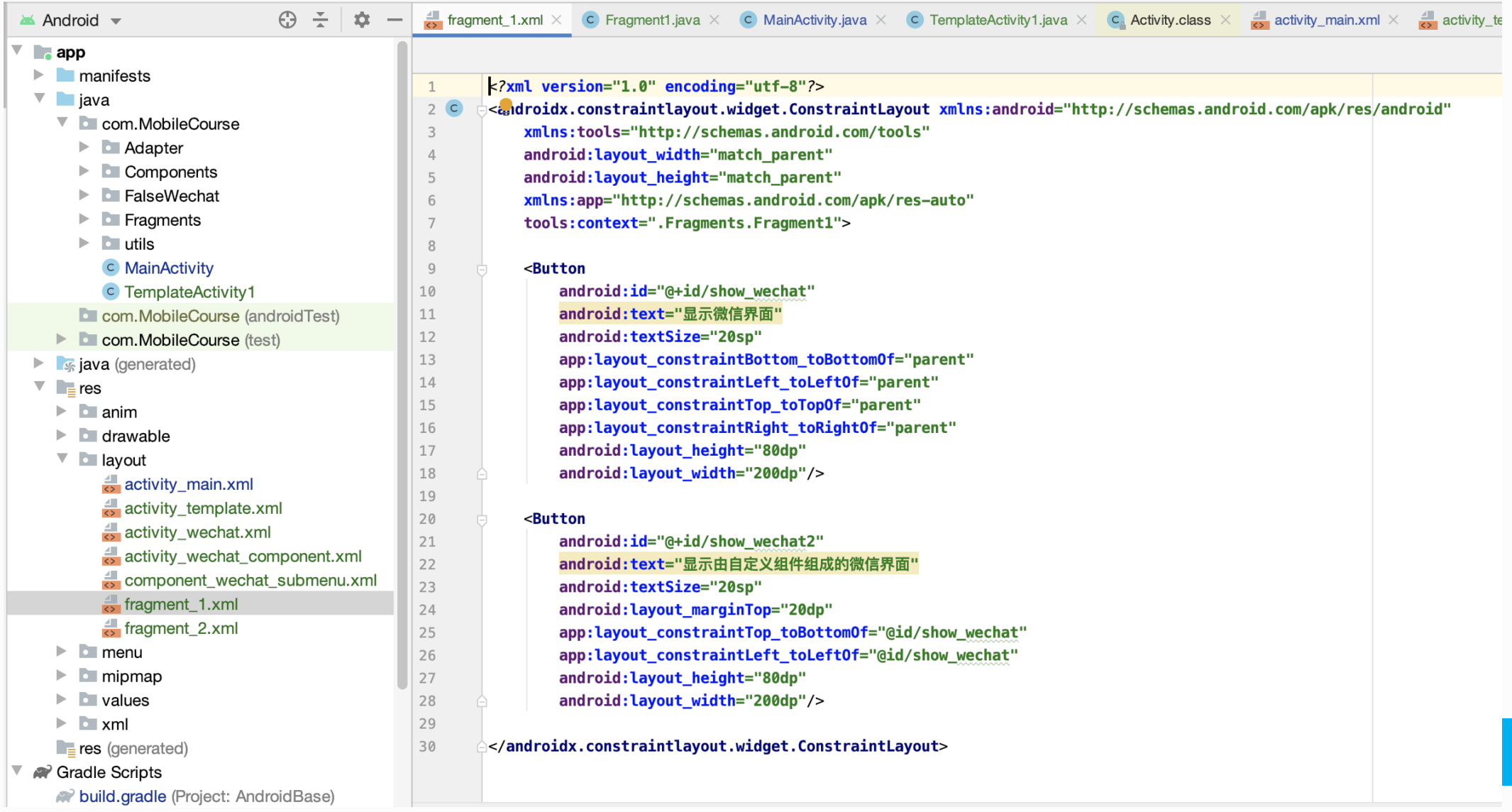
- 创建的时候调用：onAttach()，onCreate()，onCreateView()，on ActivityCreated()
- 对用户可见的时候调用：onStart() ,onResume()
- 进入后台模式调用：onPause(),onStop()
- 被销毁或者是持有它的Activity被销毁了调用：onPause() ,onStop(), onDestroyView()，onDestroy() onDetach()



使用Fragment的步骤

- 首先决定在activity中需要使用多少个fragment。例如，我们需要使用两个fragment来处理设备的横屏和竖屏两种模式。
- 下一步，基于fragment数量，创建继承自类Fragment的类。类Fragment包含回调函数。根据需求重写任意的方法。
- 对应每个片段，需要在XML文件中创建布局。这些文件中包含已定义的fragment的布局。
- 最后基于需求修改activity文件来定义实际的fragment替换逻辑。在需要加载Fragment的Activity中静态/动态加载布局文件等。

举例：在layout文件夹中创建Fragment布局文件

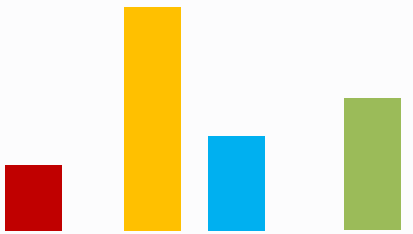


上图具体代码

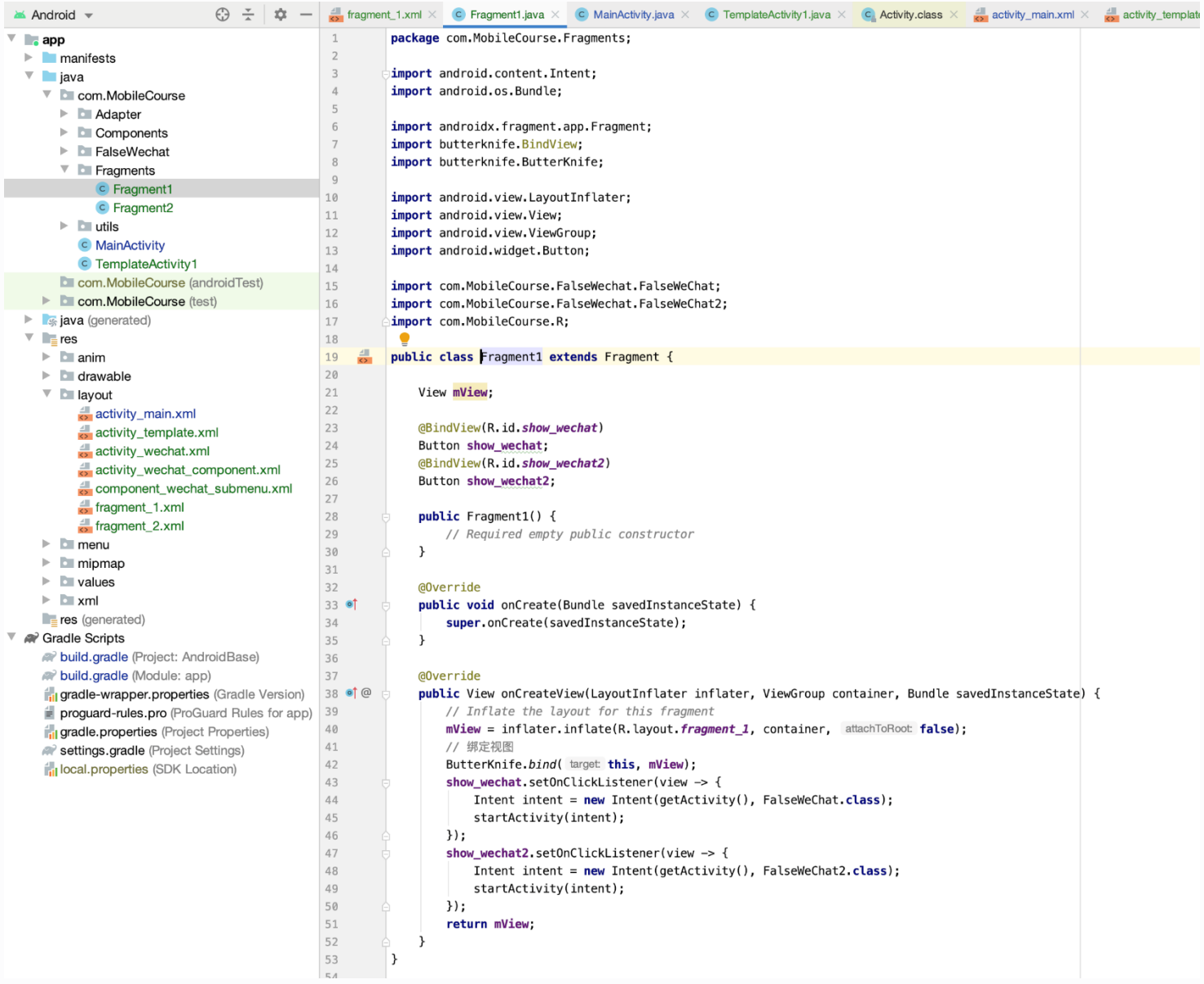
```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".Fragments.Fragment1">
    <Button ……/>
    <Button ……/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

创建继承Fragment的类

- 类名和布局文件一致FragmentID
- 重写onCreateView函数
- @Override
  public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle savedInstanceState){
  View view = inflater.inflate(R.layout. **fragmentID**, container, **false**);
  return view;
  }

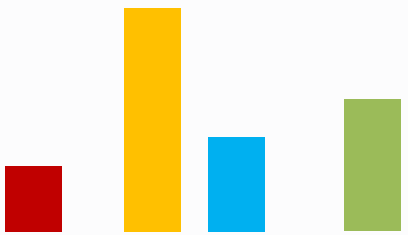


举例：继承自Fragment类，然后渲染视图



上图具体代码

```
public class Fragment1 extends Fragment {
    ......
    public Fragment1() { // Required empty public constructor }
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }
    public View onCreateView(LayoutInflater inflater,  ViewGroup container, Bundle savedInstanceState) {
      mView = inflater.inflate(R.layout.fragment_1, container, false); // Inflate the layout for this fragment
      ButterKnife.bind(this, mView); // 绑定视图
      show_wechat.setOnClickListener(view->{Intent intent=new Intent(getActivity(), FalseWeChat.class);
      startActivity(intent);});
      show_wechat2.setOnClickListener(……);
      return mView;
    }
```



静态加载

- 在需要加载Fragment的布局文件中
  - <fragment
    android:id="@+id/fragmentID" // id必须要有
    android:name="com.xx.testactivity.fragment.FragmentID"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"/>

动态加载

- 获取FragmentManager
  - FragmentManager manager = getSupportFragmentManager();
- 获取Transaction
  - FragmentTransaction transaction = manager.beginTransaction();
- 使用另一个Fragment替换当前的
  - transaction.replace(R.id.container_id, new FragmentID());
- 添加，删除，隐藏，显示，提交
  - transaction.add()
  - transaction.remove()
  - transaction.hide()
  - transaction.show()
  - transaction.commit()

举例：TemplateActivity1

- 获取Fragment管理组件，然后新建管理事务，展示或者隐藏Fragment

```
FragmentManager fm = getSupportFragmentManager();

Fragment fragment1 = new Fragment1();
Fragment fragment2 = new Fragment2( showBtn: false);
FragmentTransaction transaction = fm.beginTransaction();
transaction.add(R.id.content, fragment1);
transaction.add(R.id.content, fragment2);
transaction.show(fragment1).hide(fragment2).commit();
```

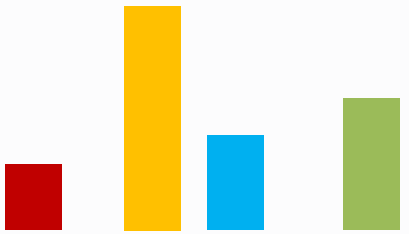
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <RelativeLayout
        android:id="@+id/content"
        android:layout_above="@id/navigation"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/navigation"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_gravity="bottom"
        android:layout_alignParentBottom="true"
        android:background="?android:attr/windowBackground"
        app:menu="@menu/navigation"
        tools:ignore="MissingConstraints" />

</RelativeLayout>
```

举例：MainActivity

- 使用ViewPager来连接Fragment，它能够方便的管理每个页面的生命周期，ViewPager通过适配器管理Fragment，常用的适配器有FragmentPagerAdapter和FragmentStatePagerAdapter。

```
FragmentManager fm = getSupportFragmentManager();
List<Fragment> fragments = new ArrayList<>();
fragments.add(new Fragment1());
fragments.add(new Fragment2());
viewPager.setAdapter(new MyFragmentAdapter(fm, fragments));
```

简易开发



简易开发

- 通过butterknife快速绑定View替代findviewbyID
- 通过封装view组件的方式，使用通用模板完成微信页面多个相同样式的layout，避免重复创建。
- 自行编写可以重复使用的类CommonInterface，对重复操作onclick进行统一管理。

通过butterknife快速绑定View

- 引入butterknife

```
implementation("com.jakewharton:butterknife:10.2.1")
annotationProcessor("com.jakewharton:butterknife-compiler:10.2.1")
```

- 通过注解绑定View

```
@BindView(R.id.viewpager)
ViewPager viewPager;
@BindView(R.id.navigation)
BottomNavigationView navigationMenu;
```

- 在设置视图文件时统一绑定

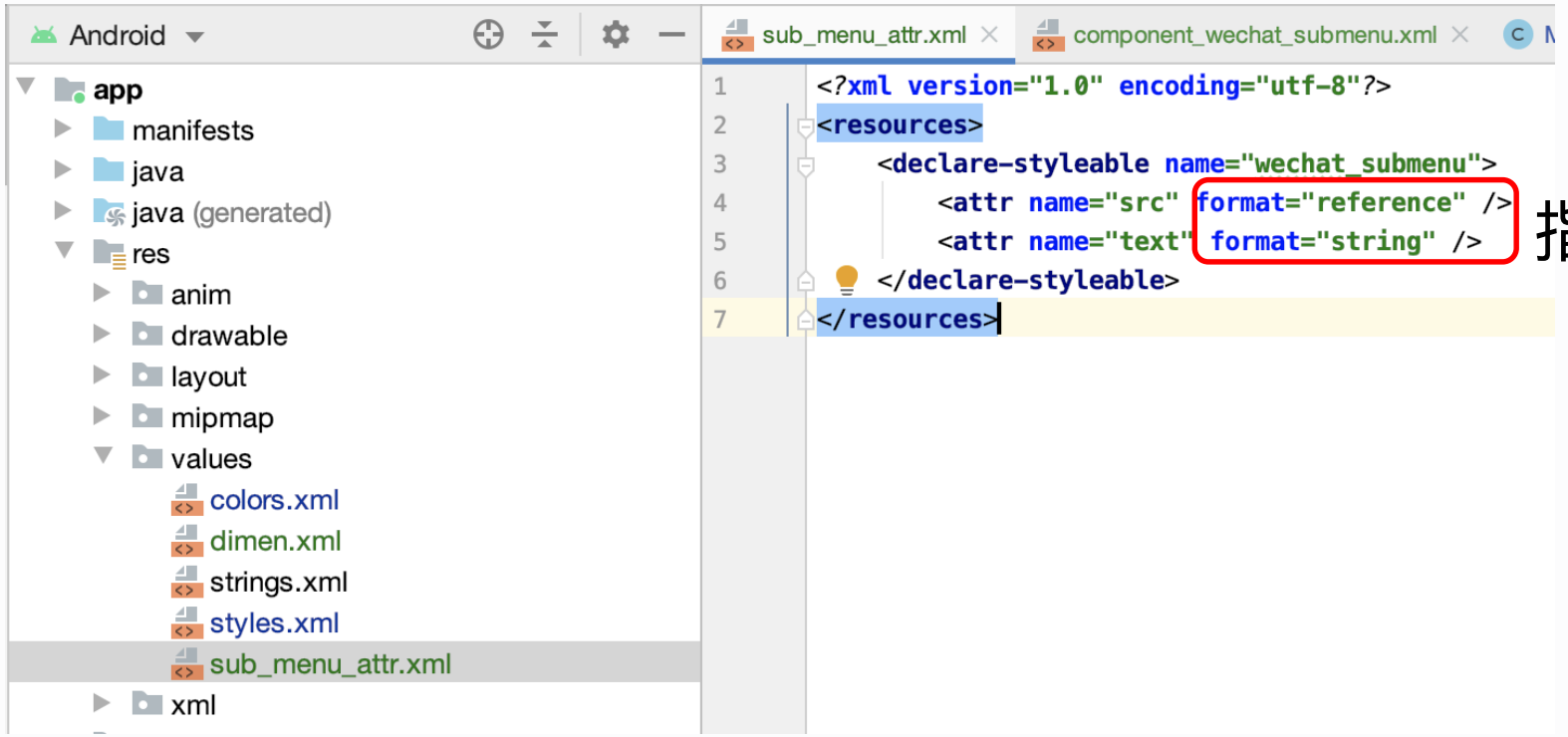
```
ButterKnife.bind(this);
```

封装View的步骤

- 在 styles.xml 文件中添加内容进行定义
- 新建一个类将 “定义”与“组件”联系起来
- 集成封装的 xml 布局
- 根据定义设置对应的值
- 使用封装的组件

封装View组件：定义

- 在values中新建xml属性文件(sub_menu_attr.xml)，此处属性名为wechat_submenu，子属性为src和text



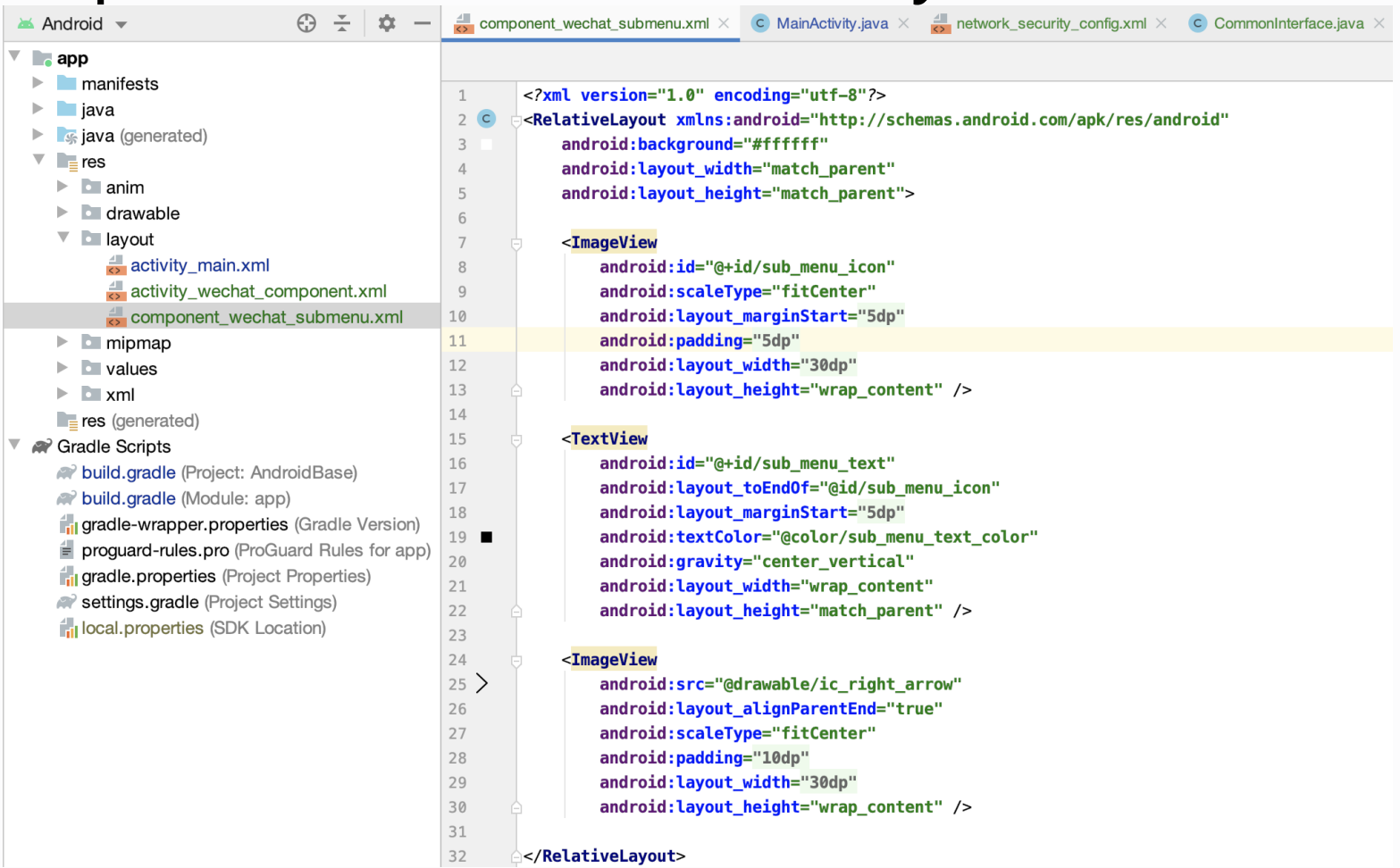
封装View组件：新建类

- 新建WechatSubmenu.java类

```
private void init(Context context, AttributeSet attrs) {
        mView = LayoutInflater.from(context).inflate
                (R.layout.component_wechat_submenu,this,true); //封装的布局文件
        ButterKnife.bind(mView);//进行快速绑定
        TypedArray a = context.obtainStyledAttributes(attrs,
                R.styleable.wechat_submenu); //得到定义的类型名
        sub_menu_icon.setImageResource(a.getResourceId
                (R.styleable.wechat_submenu_src, R.drawable.ic_social_circle));
        //拿到传进来的定义值并进行赋值
        sub_menu_text.setText(a.getString(R.styleable.wechat_submenu_text));
    }
```

封装View组件：集成封装

- 在layout中新建布局文件component_wechat_submenu.xml
- 推荐component开头，以区分activity



封装View组件：设置属性

- 在布局文件中使用自定的组件，并可以设置对应的自定义属性

```
<com.MobileCourse.Components.WechatSubmenu
    android:id="@+id/social_circle_menu"
    app:src="@drawable/ic_social_circle"
    app:text="朋友圈"
    app:layout_constraintTop_toBottomOf="@id/top_panel"
    android:layout_width="match_parent"
    android:layout_height="45dp" />
```

这两个属性是自定义的

android,app,tools的区别

- **android**
  - xmlns:android= “http://schemas.android.com/apk/res/android”
  - 有了他，Android Studio就会在我们编写布局文件的时候给出提示，提示我们可以输入什么，不可以输入什么。也可以理解为语法文件，或者语法判断器。
  - 是系统自带的公共属性。
- **app**
- **tools**

android,app,tools的区别

- **android**
- **app**
  - xmlns:app=“http://schemas.android.com/apk/res-auto”
  - 在项目需求中，我们很可能需要导入自定义控件的一些属性，或者support支持包之类的。可以将res-auto换成你的应用程序包路径，将其导入。但因为res-auto可以引用所有的自定义包名。普遍使用后者。
  - 是引入的控件特有的属性。
- **tools**

android,app,tools的区别

- **android**
- **app**
- **tools**
  - xmlns:tools= “http://schemas.android.com/tools”
  - tools会告诉安卓，哪些属性在运行时是不考虑进去的。也就是说想在运行的时候让某些属性不起作用。
  - 优点一方面为了在数据没有来的时候可以在xml看到控件位置，便于调试。二是如果网络延迟或者出错，控件值的显示为空，如果用android，就会显示你现在随意写的东西，出现bug。

封装View组件：效果展示

- 对比右图未进行组件封装的实现代码来看，左图的代码简洁程度也有了显著的提升

```
<com.MobileCourse.Components.WechatSubmenu
    android:id="@+id/people_nearby_menu"
    app:src="@drawable/ic_people_nearby"
    app:text="附近的人"
    app:layout_constraintTop_toBottomOf="@id/search_menu"
    android:background="#ffffff"
    android:layout_marginTop="20dp"
    android:layout_width="match_parent"
    android:layout_height="45dp" />
```

```
<RelativeLayout
    android:id="@+id/people_nearby_menu"
    app:layout_constraintTop_toBottomOf="@id/search_menu"
    android:background="#ffffff"
    android:layout_marginTop="20dp"
    android:layout_width="match_parent"
    android:layout_height="45dp">

    <ImageView
        android:id="@+id/img_people_nearby"
        android:src="@drawable/ic_people_nearby"
        android:scaleType="fitCenter"
        android:layout_marginStart="5dp"
        android:padding="5dp"
        android:layout_width="30dp"
        android:layout_height="wrap_content" />

    <TextView
        android:layout_toEndOf="@id/img_people_nearby"
        android:text="附近的人"
        android:layout_marginStart="5dp"
        android:textColor="@color/sub_menu_text_color"
        android:gravity="center_vertical"
        android:layout_width="wrap_content"
        android:layout_height="match_parent" />

    <ImageView
        android:src="@drawable/ic_right_arrow"
        android:layout_alignParentEnd="true"
        android:scaleType="fitCenter"
        android:layout_marginStart="5dp"
        android:padding="10dp"
        android:layout_width="30dp"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

自行编写可以简化操作的类

- 利用utils中自行编写的类进行封装好的view组件的点击事件的监听

```
CommonInterface.addViewsListener(this, new int[]{R.id.xx,…}, this);
```

- 点击后触发响应，显示不同的view id

```
public void onClick(View v) {
    Toast.makeText(v.getContext(), v.getId() + "", Toast.LENGTH_SHORT).show();
}
```


Thank
you