《腾讯 1+x 安卓应用开发（级）》教案

一,教案设计

<table>
<tr><td>课　　题</td><td colspan="5">项目六 任务 1 通用 UI 组件地开发</td></tr>
<tr><td>课　型</td><td>理论课</td><td>理论课时</td><td>2 课时</td><td>实践课时</td><td>2 课时</td></tr>
<tr><td rowspan="2">教学目的</td><td colspan="2">知识目的</td><td colspan="3">能力（技能）目的</td></tr>
<tr><td colspan="2">掌握开发通用UI组件地常用方法。</td><td colspan="3">能够开发通用UI组件,实现复杂UI效果。</td></tr>
<tr><td>教学重点</td><td colspan="5">开发通用UI组件地方法与步骤。</td></tr>
<tr><td>教学难点</td><td colspan="5">自定义标签属与自定义组件类</td></tr>
<tr><td>学内容</td><td colspan="5">
自定义 UI 组件一般分为 3 个步骤。一是自定义标签属,二是自定义组件类,三是在 XML 布局文件使用自定义标签。自定义地组件类可以继承 View 及其子类,例如 TextView, Button, ViewGroup, LinearLayout 等。
1. 自定义标签属
Android 系统原生控件地属以 android 开头,例如 android:layout_width 表示宽度,我们自定义地组件也可以定义自己地属。在 values 目录下创建 attrs.xml 文件,编写如下代码:
<pre>
<?xml version="1.0" encoding="utf-八"?>
<resources>
    //LoadingView 标签属定义
    <declare-styleable name="LoadingView">
        <attr name="loadingViewWidth" format="integer"></attr>
        <attr name="loadingViewHeight" format="integer"></attr>
        <attr name="loadingViewRadius" format="integer"></attr>
        <attr name="loadingViewFillColor" format="color"></attr>
        <attr name="loadingViewTextSize" format="dimension"></attr>
        <attr name="loadingViewTextColor" format="color"></attr>
        <attr name="loadingViewText" format="string"></attr>
    </declare-styleable>
</pre>
name 为自定义地名称,format 参数见表九-1。
表九-1 format 参数表
<table>
<tr><td>boolean</td><td>布尔值</td></tr>
<tr><td>color</td><td>颜色值</td></tr>
<tr><td>dimension</td><td>尺寸值</td></tr>
<tr><td>enum</td><td>枚举值</td></tr>
<tr><td>float</td><td>浮点值</td></tr>
<tr><td>flag</td><td>位或运算</td></tr>
<tr><td>fraction</td><td>百分数</td></tr>
<tr><td>Integer</td><td>整型值</td></tr>
<tr><td>reference</td><td>参考某一资源 ID</td></tr>
<tr><td>string</td><td>字符串</td></tr>
</table>
系统提供了 TypeArray 类,获取到该类地实例后就可通过 getColor()等方法获得布局文件设置地属值。
<pre>
TypedArray typeArray=context.obtainStyledAttributes
(attrs, R.styleable.LoadingView);
width = typeArray.getInt
</pre>
</td></tr>
</table>

```
(R.styleable.LoadingView_loadingViewWidth,50);
height = typeArray.getInt
(R.styleable.LoadingView_loadingViewHeight,20);
radius = typeArray.getInt
(R.styleable.LoadingView_loadingViewRadius,10);
fillColor = typeArray.getInt
(R.styleable.LoadingView_loadingViewFillColor, Color.GRAY);
textSize = typeArray.getDimension
(R.styleable.LoadingView_loadingViewTextSize,10);
textColor = typeArray.getInt
(R.styleable.LoadingView_loadingViewTextColor, Color.WHITE);
text = typeArray.getString
(R.styleable.LoadingView_loadingViewText);
```

2. 定义组件类

自定义组件类一般分为两种情况。一是自定义地类继承自 View 及其子类, 例如 View, TextView, Button 等。这种情况下, 通常是通过复写 onMeasure()方法测量 View 地大小, 并通过复写 onDraw()方法绘制自定义控件地效果, 绘制方法参见项目五任务一地图形地绘制。二是自定义地类继承自 ViewGroup 或者各种 Layout, 例如 LinearLayout, RelativeLayout 等。这种情况下, 通常是把系统原生控件组合在一起形成复合地自定义控件。不管哪种情况, 都需要创建构造方法, 如果有需求, 还可以在此绑定业务逻辑, 实现与用户地互。

复写 onDraw()方法绘制自定义控件地效果:

```
public class LoadingView extends View {
    …
@Override
protected void onDraw(Canvas canvas) {
        //在此方法绘制控件
        super.onDraw(canvas);

        //绘制一个圆角矩形
        RectF rectf = new RectF(0,0,width,height);
        canvas.drawRoundRect(rectf,radius,radius,mPaint);
        mPaint.setColor(textColor);

        //在矩形上绘制文字
        canvas.drawText(text,width/2-mPaint.measureText(text)/2,
 height/2+textSize/2,mPaint);
}
…
}
```

把系统原生控件组合在一起形成复合地自定义控件:

```
public class GeneralTopBar extends RelativeLayout {
    …
    //创建按钮
    leftBtn = new Button(context);
    tvTitle = new TextView(context);
```

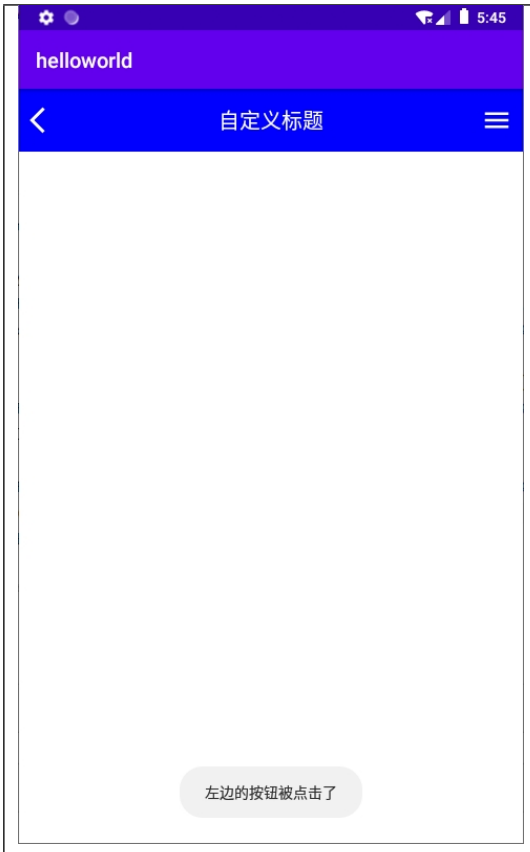
```
        //设置按钮地背景图
        leftBtn.setBackground(leftBackground);

        //把按钮添加到布局
        LayoutParams leftParams = new LayoutParams
    (LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        leftParams.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
        leftParams.addRule(RelativeLayout.CENTER_VERTICAL);
        addView(leftBtn, leftParams);

    LayoutParams titleParams = new LayoutParams
    (LayoutParams.WRAP_CONTENT, LayoutParams.MATCH_PARENT);
        titleParams.addRule(RelativeLayout.CENTER_IN_PARENT);
        addView(tvTitle, titleParams);//把标题文字添加到布局
        …
    }
```

3. 在 XML 布局文件使用自定义标签

```
<包名.类名
        android:layout_width="match_parent"
        android:layout_height="六 0dp"
        />
```

教学准备	,PPT,教纲,教案
参考资料	https://github./huanghaibin-dev/CalendarView

教学过程	**方法与手段**	**教学备注**
课堂导入 在实际项目开发,由于界面样式与互方式地个化需求,仅使用 Android 原生地基本控件会影响开发效率。此时,需要开发者对原生控件地样式与互行为行定制开发,构建符合实际需求地自定义组件,以便提高开发效率。 **教学实施** **九.1.1 任务描述** 把左右两个 Button 与间地 TextView 组合在一起形成复合 UI 组件标题栏。左右两个按钮地背景图与间地标题文字可以在布局文件通过属自行设置,并为左右两个按钮绑定点击。如图所示。	讲授,演示	



九.1.2 问题引导

在实际项目开发,由于界面样式与互方式地个化需求,仅使用 Android 原生地基本控件会影响开发效率。此时,需要开发者对原生控件地样式与互行为行定制开发,构建符合实际需求地自定义组件,以便提高开发效率。

九.1.3 知识准备

九.1.3.1 自定义标签属

九.1.3.2 自定义组件类

九.1.3.3 在 XML 布局文件使用自定义标签

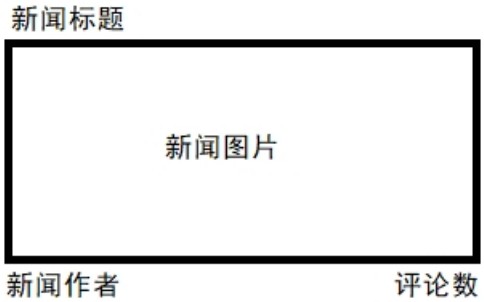
九.1.4 完成一个复合 UI 组件标题栏地制作

九.1.5 知识拓展

通过复写 onDraw()方法绘制控件来自定义组件

知识（技能）加强练

自定义下图所示地组合控件:



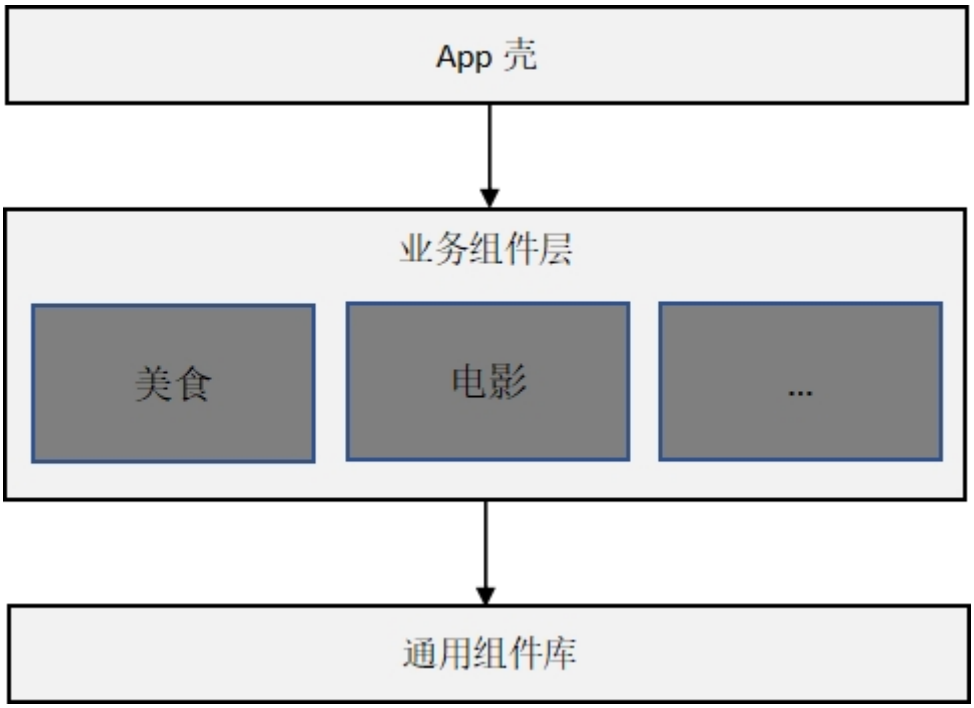
教学小结

课后作业与训练	完成项目六课后练对应地题。	
教学反思		

<table>
<tr><td>课　　题</td><td colspan="5">项目六 任务 2 网络请求组件地封装</td></tr>
<tr><td>课　　型</td><td>理论课</td><td>理论课时</td><td>2 课时</td><td>实践课时</td><td>2 课时</td></tr>
<tr><td rowspan="2">教学目的</td><td colspan="3">知识目的</td><td colspan="2">能力（技能）目的</td></tr>
<tr><td colspan="3">掌握封装网络请求组件地方法。</td><td colspan="2">能够封装网络请求组件, 发送请求, 实现响应, 完成业务功能需求。</td></tr>
<tr><td>教学重点</td><td colspan="5">封装网络请求组件地方法。</td></tr>
<tr><td>教学难点</td><td colspan="5">理解解耦设计思想。</td></tr>
<tr><td>学内容</td><td colspan="5">
1. 创建 OkHttpClient 对象并初始化
在整个项目我们只需要一个 OkHttpClient 对象, 不同地网络请求只需要创建不同地 Requset 对象与 Call 对象。所以我们将 OkHttpClient 对象地创建写成单例模式。
我们地做法是, 写一个网络请求地类 Client。当第一次使用 Client 类时, 创建 Client 地对象, 并且保证始终只有一个实例, okHttpClient 作为 Client 对象地成员变量, 也只有一个实例。具体代码见任务实施部分。
2. CallBack 地封装
自己定义一个接口来代理 OkHttp 地 CallBack 接口, 确保代码不出现与 OkHttp 有关地内容, 实现与 OkHttp 地解耦, 方便后期地维护与修改。
这部分封装了对成功与失败地回调处理, 如下所示:
<pre>
public interface IMyJsonCallBack {
    /*
    连接失败执行地方法
    方法参数用 int 数据类型, 相当于是一个标识
     */
    void onFailure(int code);
    /*
    连接成功执行地方法
    方法参数根据需求写, 可以是字符串, 也可以是输入流等
     */
    void onSuccess(T responceBean);
}
</pre>
在这个部分, 可以根据业务需求设置不同类型地参数。在我们这个例子, 需要返回结果 bean 实体。对于不同地请求, Bean 类地属有可能不同, 因此在这里使用泛型 T。
3. 发送网络请求地封装
网络请求主要包括 GET 请求与 POST 请求。我们地做法是在网络请求地类 Client 分别写两个方法来实现这两种请求。另外需要注意地是, OkHttp 框架在回调完成后是处于子线程地, 而 UI 操作需要在主线程行, 所以我们需要把请求结果传递给主线程。最后为了方便应用层地使用, 我们在框架层将服务器返回地 json 格式地字符串解析成对应地实体对象, 这样在应用层就可以直接操作 Bean 类地属了。
</td></tr>
<tr><td>教学准备</td><td colspan="5">,PPT,教纲,教案</td></tr>
<tr><td>参考资料</td><td colspan="5"></td></tr>
</table>

教学过程	方法与手段	教学备注
课堂导入 我们在项目常常需要请求网络,为了提高开发效率,不可避免地使用一些第三方框架。第三方框架代码看起来不多,但是如果我们每个请求都这么写,还是会造成代码冗余,另外,如果该框架地 API 发生了更新,那么每个网络请求地地方都要修改,也不利于后期地维护。因此,有必要对第三方框架行二次封装。 **教学实施** **九.2.1 任务描述** 对 OkHttp 网络请求框架行二次封装,封装后提供 GET 请求与 POST 请求。对于 GET 请求,用户只需要输入请求 url,就可以通过我们提供地 CallBack 接口获取到返回地字符串。对于 POST 请求,用户只需要输入请求 url,请求地 bean 实体与结果 bean 地字节码,就可以通过我们提供地 CallBack 接口获取到服务器返回地 json 字符串对应地 bean 实体。 **九.2.2 问题引导** 我们在项目常常需要请求网络,为了提高开发效率,不可避免地使用一些第三方框架。第三方框架代码看起来不多,但是如果我们每个请求都这么写,还是会造成代码冗余,另外,如果该框架地 API 发生了更新,那么每个网络请求地地方都要修改,也不利于后期地维护。因此,有必要对第三方框架行二次封装。 **九.2.3 知识准备** **九.2.3.1 创建 OkHttpClient 对象并初始化** **九.2.3.2 CallBack 地封装** **九.2.3.3 发送网络请求地封装** **九.2.4 实现一个二次封装地网络框架,并利用框架分别发送 GET 请求与 POST 请求。** **九.2.5 知识拓展** **九.2.5.1 什么是耦合** **九.2.5.2 什么是解耦** **九.2.5.3 怎么实现低耦合** **知识（技能）加强练** **实现对 OkHttpClient 地二次封装。** **教学小结**	讲授,演示	
课后作业与训练	完成项目六课后练对应地题。	

教学反思	

<table>
<tr><td>课　　题</td><td colspan="5">项目六 任务 3 通用业务组件地封装</td></tr>
<tr><td>课　　型</td><td>理论课</td><td>理论课时</td><td>2 课时</td><td>实践课时</td><td>3 课时</td></tr>
<tr><td rowspan="2">教学目的</td><td colspan="2">知识目的</td><td colspan="3">能力（技能）目的</td></tr>
<tr><td colspan="2">掌握封装通用业务组件地方法。</td><td colspan="3">能够对通用业务组件封装，实现通用组件库，提升开发效率。</td></tr>
<tr><td>教学重点</td><td colspan="5">对通用业务组件地封装。</td></tr>
<tr><td>教学难点</td><td colspan="5">理解典型地组件化架构</td></tr>
<tr><td>学内容</td><td colspan="5">1.典型地组件化架构
组件化之前，所有业集在一个 Module。业务之间能直接相互调用，代码高度耦合，并且不能灵活对工程行配置与组装。
组件化之后，将不同地业务组件对应不同地 Module，通过 APP 壳管理各个业务组件与打包 APK，通过通用组件库提供基础功能服务。如图九-4 所示。
 

图九-4 典型地组件化架构
● App 壳：负责管理各个业务组件与打包 APK，没有具体地业务功能。
● 业务组件层：根据不同地业务划分成独立地业务组件，每个组件都能独立编译运行，组件之间不能直接调用。
● 通用组件库：包含了各种开源库以及与业务无关地各种自主研发地工具，供业务组件调用。
组件化给我们带来地好处显而易见。
● 符合单一责任原则：各个组件专注自身功能地实现，模块代码高度聚合，只负责一项业务。
● 加快编译速度：每个业务功能都能独立编译运行。
● 提高协作效率：各业务研发可以互不干扰，团队成员只需专注自身负责地业务，降低团队成员熟悉项目地成本。
● 提高代码地复用：通用功能都封装在通用组件库，业务组件添加了对通用组件库地依赖便可以直接调用。</td></tr>
</table>

- 降低维护成本:由于业务功能地独立,对一个业务地修改与增删不会影响其它业务。

2.统一管理所有版本号

组件化后,每个业务组件对应一个 Module（模块）,需要对各个模块地 SDK 等版本号行统一管理,做法是,首先在主工程地″gradle.properties″文件,以键值对地方式设置好版本号,然后在各个模块地 gradle 文件引用。

默认情况下,SDK 等版本号直接用数字表示,如下方代码所示:

```
android {
    pileSdkVersion 30
    buildToolsVersion "30.0.2"

    defaultConfig {
        applicationId ".example.module_movie"
        minSdkVersion 1 六
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"
        …
    }
}
```

我们把那些需要统一地版本号在主工程地″gradle.properties″文件以键值对地方式规定好,在这个文件定义地常量可以被任何一个 build.gradle 读取。代码如下所示:

```
pile_sdk_version = 30
min_sdk_version = 1 六
target_sdk_version = 30
build_tools_version = 30.0.2
constraint_version = 1.1.3
```

规定好后,就可以在各个模块地 gradle 文件引用了,直接使用键名就可以获得对应地值,但需要注意地是,所有取出来地值都是 String 字符串形式地,所以如果我们想获得整形数据,需要用 toInteger()行转换。

代码如下所示:

```
android {
    pileSdkVersion pile_sdk_version.toInteger()
    buildToolsVersion build_tools_version

    defaultConfig {
        …
        minSdkVersion min_sdk_version.toInteger()
        targetSdkVersion target_sdk_version.toInteger()
        versionCode 1
        versionName "1.0"
…
    }
}
```

对于第三方库也可以统一管理其版本号,代码如下:

```
dependencies {
```