电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 李 欣 芮

姓 名 李 欣 芮

（实验） 课程名称**Android系统结构与应用编程**

理论教师 饶 云 波

实验教师 饶 云 波

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：李欣芮 学号：201822090530 指导教师：饶云波**

**实验地点：电子科技大学 实验时间：2019.5.5**

**一、实验名称：Android 界面布局及基本控件**

**二、实验学时：4学时**

**三、实验目的：**

* 熟悉Android Studio3.0开发软件
* 熟悉Android的基本控件；
* 熟悉Android的界面布局；

**四、实验原理：**

Android Studio3.0 开发软件；

Android基本控件和界面布局；

**五、实验内容：**

主要完成以下基本控件和界面布局的调试。

* 线性布局 Linear layout
* 滚动视图 ScrollView
* 文本视图 TextView
* 按钮 Button
* 图像视图 ImageView
* 图像按钮 ImageButton

1.实战项目：简单计算器

* 相对布局 RelativeLayout
* 框架布局 FrameLayout
* 编辑框 EditText
* 特殊按钮 CheckBox, Switch, RadioButton
* 日期时间控件 DatePicker, TimePicker
* 列表视图 ListView

2.项目实战: 房贷计算器

**六、实验器材（设备、元器件）：**

PC机，Android Studio3.0软件，JDK包，MySql数据库软件等。

**七、实验步骤：**

**1．创建空项目**

**2. 创建layout布局文件**

**3. 创建main\_activity文件，将事件写入**

**4. 调试该项目**

**5. 发布并创建APP**

**6. 同样的步骤建立房贷计算器。**

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

**MainActivity**

package com.example.simplecaculator;  
  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.Button;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import com.example.simplecaculator.utils.Caculator;  
import com.googlecode.aviator.AviatorEvaluator;  
  
public class MainActivity extends AppCompatActivity {  
  
 private TextView tv\_exp;  
 private TextView tv\_res;  
 private String exp;  
 private String res;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 tv\_exp = findViewById(R.id.*tv\_exp*);  
 tv\_res = findViewById(R.id.*tv\_res*);  
 exp = "";  
 res = "";  
 }  
  
 public void numClick(View view) {  
 if(!res.equals("")){  
 Log.*d*("MainActivity", "res = "+res);  
 exp = "";  
 res = "";  
 tv\_exp.setText(exp);  
 tv\_res.setText(res);  
 }else Log.*d*("MainActivity", "res = null");  
 Button btn = findViewById(view.getId());  
 exp = exp + btn.getText();  
 tv\_exp.setText(exp);  
 }  
  
 public void equalClick(View view) {  
 Log.*d*("MainActivity", "exp = "+exp);  
 if(exp.equals("")) Toast.makeText(getApplicationContext(), "请先输入表达式!", Toast.*LENGTH\_SHORT*).show();  
 try {  
 Log.*d*("MainActivity", "res.type = "+AviatorEvaluator.*execute*(exp).getClass().getSimpleName());  
 double temp = Caculator.*conversion*(exp);  
 res = String.*valueOf*(temp);  
 }catch (Exception e){  
 Log.*d*("MainActivity", "exp\_exception : " + e.getMessage());  
 Toast.makeText(getApplicationContext(), "请重新输入合法的表达式!", Toast.*LENGTH\_SHORT*).show();  
 }  
 tv\_res.setText(res);  
 }  
  
 public void clearClick(View view) {  
 exp = "";  
 res = "";  
 tv\_exp.setText(exp);  
 tv\_res.setText(res);  
 }  
  
  
 public void delClick(View view) {  
 if(exp.length()>0) {  
 exp = exp.substring(0, exp.length() - 1);  
 tv\_exp.setText(exp);  
 }  
  
 }  
}

用于大数精确计算

package com.example.simplecaculator.utils;  
  
public class ArithHelper {  
  
 // 默认除法运算精度  
 private static final int *DEF\_DIV\_SCALE* = 16;  
  
 // 这个类不能实例化  
 private ArithHelper() {  
 }  
  
 */\*\*  
 \* 提供精确的加法运算。  
 \*  
 \** ***@param*** *v1 被加数  
 \** ***@param*** *v2 加数  
 \** ***@return*** *两个参数的和  
 \*/* public static double add(double v1, double v2) {  
 java.math.BigDecimal b1 = new java.math.BigDecimal(Double.*toString*(v1));  
 java.math.BigDecimal b2 = new java.math.BigDecimal(Double.*toString*(v2));  
 return b1.add(b2).doubleValue();  
 }  
  
 public static double add(String v1, String v2) {  
 java.math.BigDecimal b1 = new java.math.BigDecimal(v1);  
 java.math.BigDecimal b2 = new java.math.BigDecimal(v2);  
 return b1.add(b2).doubleValue();  
 }  
  
 */\*\*  
 \* 提供精确的减法运算。  
 \*  
 \** ***@param*** *v1 被减数  
 \** ***@param*** *v2 减数  
 \** ***@return*** *两个参数的差  
 \*/* public static double sub(double v1, double v2) {  
 java.math.BigDecimal b1 = new java.math.BigDecimal(Double.*toString*(v1));  
 java.math.BigDecimal b2 = new java.math.BigDecimal(Double.*toString*(v2));  
 return b1.subtract(b2).doubleValue();  
 }  
  
 public static double sub(String v1, String v2) {  
 java.math.BigDecimal b1 = new java.math.BigDecimal(v1);  
 java.math.BigDecimal b2 = new java.math.BigDecimal(v2);  
 return b1.subtract(b2).doubleValue();  
 }  
  
 */\*\*  
 \* 提供精确的乘法运算。  
 \*  
 \** ***@param*** *v1  
 \* 被乘数  
 \** ***@param*** *v2  
 \* 乘数  
 \** ***@return*** *两个参数的积  
 \*/* public static double mul(double v1, double v2) {  
 java.math.BigDecimal b1 = new java.math.BigDecimal(Double.*toString*(v1));  
 java.math.BigDecimal b2 = new java.math.BigDecimal(Double.*toString*(v2));  
 return b1.multiply(b2).doubleValue();  
 }  
  
 public static double mul(String v1, String v2) {  
 java.math.BigDecimal b1 = new java.math.BigDecimal(v1);  
 java.math.BigDecimal b2 = new java.math.BigDecimal(v2);  
 return b1.multiply(b2).doubleValue();  
 }  
  
 */\*\*  
 \* 提供（相对）精确的除法运算，当发生除不尽的情况时，精确到 小数点以后10位，以后的数字四舍五入。  
 \*  
 \** ***@param*** *v1  
 \* 被除数  
 \** ***@param*** *v2  
 \* 除数  
 \** ***@return*** *两个参数的商  
 \*/* public static double div(double v1, double v2) {  
 return *div*(v1, v2, *DEF\_DIV\_SCALE*);  
 }  
  
 public static double div(String v1, String v2) {  
 java.math.BigDecimal b1 = new java.math.BigDecimal(v1);  
 java.math.BigDecimal b2 = new java.math.BigDecimal(v2);  
 return b1.divide(b2, *DEF\_DIV\_SCALE*, java.math.BigDecimal.*ROUND\_HALF\_UP*).doubleValue();  
 }  
  
 */\*\*  
 \* 提供（相对）精确的除法运算。当发生除不尽的情况时，由scale参数指 定精度，以后的数字四舍五入。  
 \*  
 \** ***@param*** *v1 被除数  
 \** ***@param*** *v2 除数  
 \** ***@param*** *scale 表示表示需要精确到小数点以后几位。  
 \** ***@return*** *两个参数的商  
 \*/* public static double div(double v1, double v2, int scale) {  
 if (scale < 0) {  
 throw new IllegalArgumentException("The scale must be a positive integer or zero");  
 }  
 java.math.BigDecimal b1 = new java.math.BigDecimal(Double.*toString*(v1));  
 java.math.BigDecimal b2 = new java.math.BigDecimal(Double.*toString*(v2));  
 return b1.divide(b2, scale, java.math.BigDecimal.*ROUND\_HALF\_UP*).doubleValue();  
 }  
  
 */\*\*  
 \* 提供精确的小数位四舍五入处理。  
 \*  
 \** ***@param*** *v 需要四舍五入的数字  
 \** ***@param*** *scale 小数点后保留几位  
 \** ***@return*** *四舍五入后的结果  
 \*/* public static double round(double v, int scale) {  
 if (scale < 0) {  
 throw new IllegalArgumentException("The scale must be a positive integer or zero");  
 }  
 java.math.BigDecimal b = new java.math.BigDecimal(Double.*toString*(v));  
 java.math.BigDecimal one = new java.math.BigDecimal("1");  
 return b.divide(one, scale, java.math.BigDecimal.*ROUND\_HALF\_UP*).doubleValue();  
 }  
 public static double round(String v, int scale) {  
 if (scale < 0) {  
 throw new IllegalArgumentException("The scale must be a positive integer or zero");  
 }  
 java.math.BigDecimal b = new java.math.BigDecimal(v);  
 java.math.BigDecimal one = new java.math.BigDecimal("1");  
 return b.divide(one, scale, java.math.BigDecimal.*ROUND\_HALF\_UP*).doubleValue();  
 }  
}

package com.example.simplecaculator.utils;  
import java.util.Collections;  
import java.util.Stack;  
  
*/\*\*  
 \* 算数表达式求值  
 \* 直接调用Calculator的类方法conversion()  
 \* 传入算数表达式，将返回一个浮点值结果  
 \* 如果计算过程错误，将返回一个NaN  
 \*/*public class Caculator {  
 private Stack<String> postfixStack = new Stack<String>();// 后缀式栈  
 private Stack<Character> opStack = new Stack<Character>();// 运算符栈  
 private int[] operatPriority = new int[] { 0, 3, 2, 1, -1, 1, 0, 2 };// 运用运算符ASCII码-40做索引的运算符优先级  
  
 public static double conversion(String expression) {  
 double result = 0;  
 Caculator cal = new Caculator();  
 try {  
 expression = *transform*(expression);  
 result = cal.calculate(expression);  
 } catch (Exception e) {  
 // e.printStackTrace();  
 // 运算错误返回NaN  
 return 0.0 / 0.0;  
 }  
 // return new String().valueOf(result);  
 return result;  
 }  
  
 */\*\*  
 \* 将表达式中负数的符号更改  
 \*  
 \** ***@param*** *expression  
 \* 例如-2+-1\*(-3E-2)-(-1) 被转为 ~2+~1\*(~3E~2)-(~1)  
 \** ***@return*** *\*/* private static String transform(String expression) {  
 char[] arr = expression.toCharArray();  
 for (int i = 0; i < arr.length; i++) {  
 if (arr[i] == '-') {  
 if (i == 0) {  
 arr[i] = '~';  
 } else {  
 char c = arr[i - 1];  
 if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '(' || c == 'E' || c == 'e') {  
 arr[i] = '~';  
 }  
 }  
 }  
 }  
 if(arr[0]=='~'||arr[1]=='('){  
 arr[0]='-';  
 return "0"+new String(arr);  
 }else{  
 return new String(arr);  
 }  
 }  
  
 */\*\*  
 \* 按照给定的表达式计算  
 \*  
 \** ***@param*** *expression  
 \* 要计算的表达式例如:5+12\*(3+5)/7  
 \** ***@return*** *\*/* public double calculate(String expression) {  
 Stack<String> resultStack = new Stack<String>();  
 prepare(expression);  
 Collections.*reverse*(postfixStack);// 将后缀式栈反转  
 String firstValue, secondValue, currentValue;// 参与计算的第一个值，第二个值和算术运算符  
 while (!postfixStack.isEmpty()) {  
 currentValue = postfixStack.pop();  
 if (!isOperator(currentValue.charAt(0))) {// 如果不是运算符则存入操作数栈中  
 currentValue = currentValue.replace("~", "-");  
 resultStack.push(currentValue);  
 } else {// 如果是运算符则从操作数栈中取两个值和该数值一起参与运算  
 secondValue = resultStack.pop();  
 firstValue = resultStack.pop();  
  
 // 将负数标记符改为负号  
 firstValue = firstValue.replace("~", "-");  
 secondValue = secondValue.replace("~", "-");  
  
 String tempResult = calculate(firstValue, secondValue, currentValue.charAt(0));  
 resultStack.push(tempResult);  
 }  
 }  
 return Double.*valueOf*(resultStack.pop());  
 }  
  
 */\*\*  
 \* 数据准备阶段将表达式转换成为后缀式栈  
 \*  
 \** ***@param*** *expression  
 \*/* private void prepare(String expression) {  
 opStack.push(',');// 运算符放入栈底元素逗号，此符号优先级最低  
 char[] arr = expression.toCharArray();  
 int currentIndex = 0;// 当前字符的位置  
 int count = 0;// 上次算术运算符到本次算术运算符的字符的长度便于或者之间的数值  
 char currentOp, peekOp;// 当前操作符和栈顶操作符  
 for (int i = 0; i < arr.length; i++) {  
 currentOp = arr[i];  
 if (isOperator(currentOp)) {// 如果当前字符是运算符  
 if (count > 0) {  
 postfixStack.push(new String(arr, currentIndex, count));// 取两个运算符之间的数字  
 }  
 peekOp = opStack.peek();  
 if (currentOp == ')') {// 遇到反括号则将运算符栈中的元素移除到后缀式栈中直到遇到左括号  
 while (opStack.peek() != '(') {  
 postfixStack.push(String.*valueOf*(opStack.pop()));  
 }  
 opStack.pop();  
 } else {  
 while (currentOp != '(' && peekOp != ',' && compare(currentOp, peekOp)) {  
 postfixStack.push(String.*valueOf*(opStack.pop()));  
 peekOp = opStack.peek();  
 }  
 opStack.push(currentOp);  
 }  
 count = 0;  
 currentIndex = i + 1;  
 } else {  
 count++;  
 }  
 }  
 if (count > 1 || (count == 1 && !isOperator(arr[currentIndex]))) {// 最后一个字符不是括号或者其他运算符的则加入后缀式栈中  
 postfixStack.push(new String(arr, currentIndex, count));  
 }  
  
 while (opStack.peek() != ',') {  
 postfixStack.push(String.*valueOf*(opStack.pop()));// 将操作符栈中的剩余的元素添加到后缀式栈中  
 }  
 }  
  
 */\*\*  
 \* 判断是否为算术符号  
 \*  
 \** ***@param*** *c  
 \** ***@return*** *\*/* private boolean isOperator(char c) {  
 return c == '+' || c == '-' || c == '\*' || c == '/' || c == '(' || c == ')';  
 }  
  
 */\*\*  
 \* 利用ASCII码-40做下标去算术符号优先级  
 \*  
 \** ***@param*** *cur  
 \** ***@param*** *peek  
 \** ***@return*** *\*/* public boolean compare(char cur, char peek) {// 如果是peek优先级高于cur，返回true，默认都是peek优先级要低  
 boolean result = false;  
 if (operatPriority[(peek) - 40] >= operatPriority[(cur) - 40]) {  
 result = true;  
 }  
 return result;  
 }  
  
 */\*\*  
 \* 按照给定的算术运算符做计算  
 \*  
 \** ***@param*** *firstValue  
 \** ***@param*** *secondValue  
 \** ***@param*** *currentOp  
 \** ***@return*** *\*/* private String calculate(String firstValue, String secondValue, char currentOp) {  
 String result = "";  
 switch (currentOp) {  
 case '+':  
 result = String.*valueOf*(ArithHelper.*add*(firstValue, secondValue));  
 break;  
 case '-':  
 result = String.*valueOf*(ArithHelper.*sub*(firstValue, secondValue));  
 break;  
 case '\*':  
 result = String.*valueOf*(ArithHelper.*mul*(firstValue, secondValue));  
 break;  
 case '/':  
 result = String.*valueOf*(ArithHelper.*div*(firstValue, secondValue));  
 break;  
 }  
 return result;  
 }  
}

**九、总结及心得体会：**

本次实验主要对简单计算器的APP的实现，通过对该实验的训练，熟练了对相对布局、网格布局，线性布局，并且对基本掌握了多选框，单选框，编辑框，文本view等控件的用法。

**十、对本实验过程及方法、手段的改进建议：**

**无**

**报告评分：**

**指导教师签字：**