

移动应用开发课程作业 - 计算器的设计与实现

指导老师：曹阳

2017 年 10 月 15 日

华南师范大学计算机学院

计算机科学与技术（软件技术应用方向）

2015 级 5 班

王子彦

20142100012

ziyan.wang@m.scnu.edu.cn

声明：除特殊注明外，本次作业中所有的文档、源代码和相关资源均为本人原创。

该项目的源代码已在 Github 上以 Apache-2.0 协议发布：

<https://github.com/lonelyenvoy/Calculator>

王子彦授予曹阳老师对本项目所有文件的查阅、拷贝、修改和使用的权利。

软件名称	Calculator
完成人	王子彦
学号	20142100012
完成时间	2017 年 10 月 15 日

一、 软件内容简介

本项目实现了科学计算器的功能，具有较高的综合质量，优雅的 UI 设计，出色的性能，完善的异常处理机制与友好的提示信息。

共计一名开发人员（王子彦本人）和 12 名测试人员参与了本项目，其中 12 名测试人员在产品测试过程中为开发者提供了宝贵的意见和建议，帮助开发者发现并修复了大量的 bug 和漏洞，在此王子彦对测试人员表达由衷的感谢和敬意。

截至本文发布前，本项目代码的最新版本为 1.0.10。

Github 项目 Readme 页面：

您可在[此处](#)查阅本项目的所有源码，或者用微信以外的工具扫描二维码下载 app。

lonelyenvoy / Calculator

Watch0

Star0

Fork0

<> Code

Issues0

Pull requests0

Projects0

Insights

A simple calculator for Android

10 commits

1 branch

0 releases

1 contributor

Apache-2.0

Branch: master

New pull request

Find file

Clone or download


lonelyenvoy fix bugs

Latest commit 1480ff0 2 hours ago

README.md


Calculator

A simple calculator for Android



Download

You may download the android apk file [here](#), or scan the QR code below:



License

[The Apache-2.0 License](#)

二、界面设计

本次 app 的设计中将界面分为了两部分，即显示区域和操作区域。

显示区域有两行可伸缩的文本，分别用于显示用户输入的表达式和计算结果。

操作区域使用了 5 行 * 5 列的按键矩阵，在有限的区域内提供给用户最大化的选择，允许用户随心所欲地构造计算表达式。

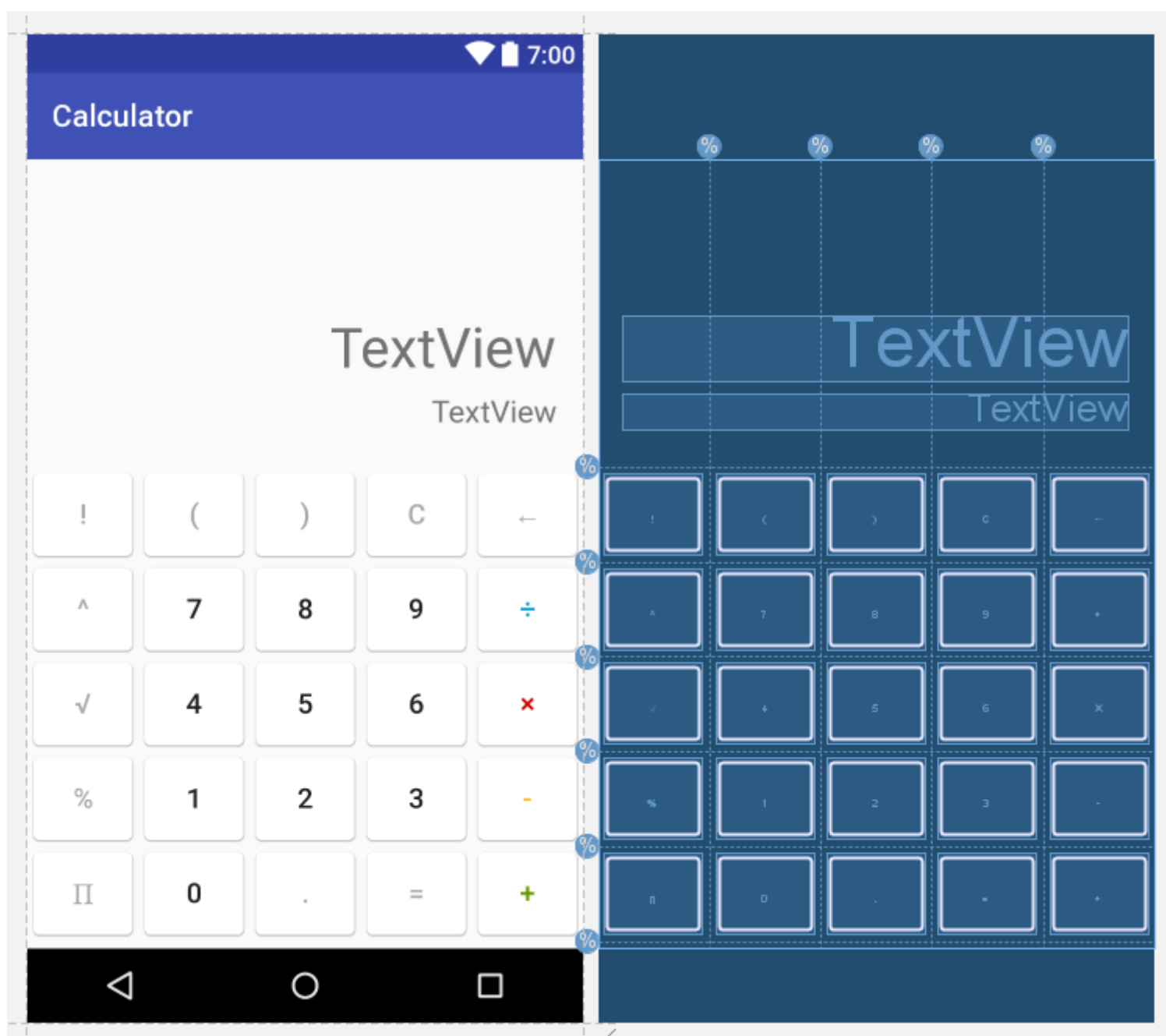


图 2-1 Android Studio 界面设计原型图

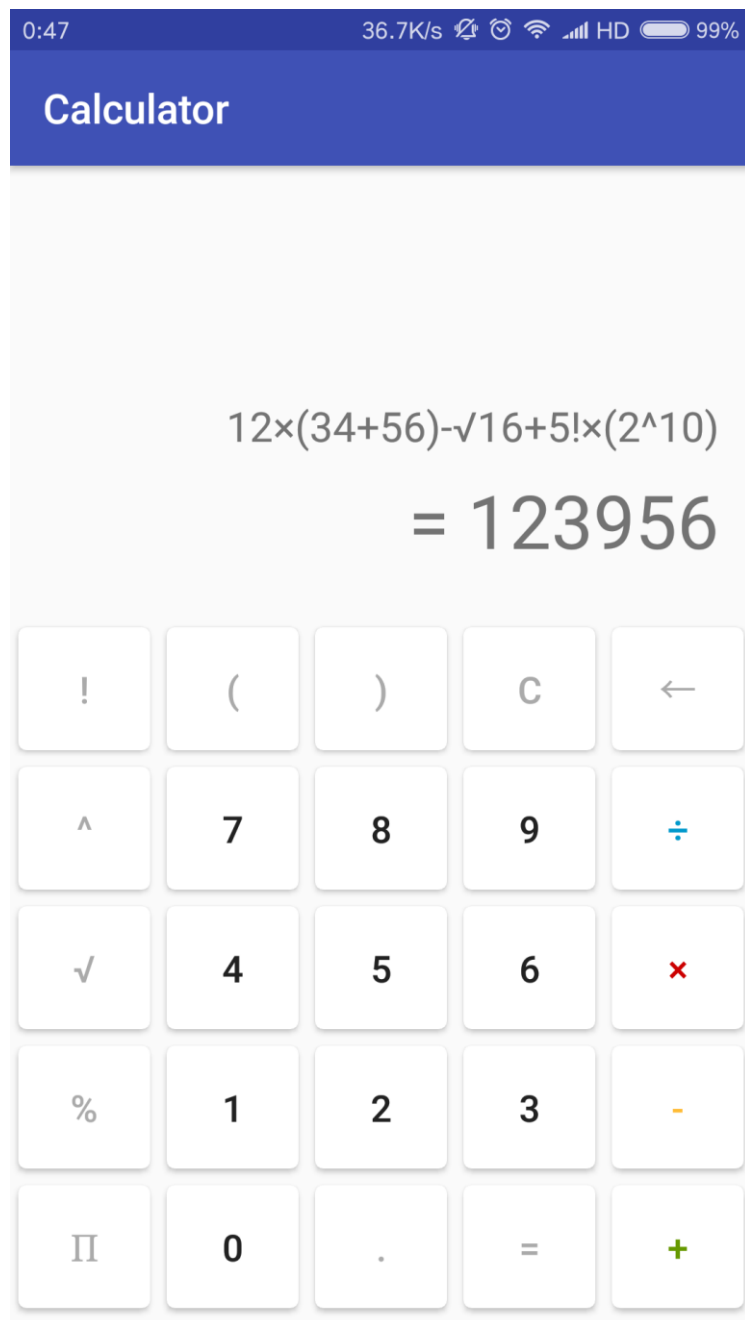


图 2-2 App 真机运行截图

为了您能更直观地了解本次作业的成果，请您[点击此处查看 App 完整功能演示（10 秒）](#)，或查看同目录下的“app 完整功能演示图.gif”文件

三、项目设计中的亮点

3.1 高级计算器

3.1.1 可扩展不定长的连续表达式识别与执行

从理论上来说, 开发者所编写的计算器能够执行无限长的表达式, 并且显示正确的运算结果, 无论表达式有多长, 用户输入的算式和计算结果仍然能够优雅地显示在屏幕上。

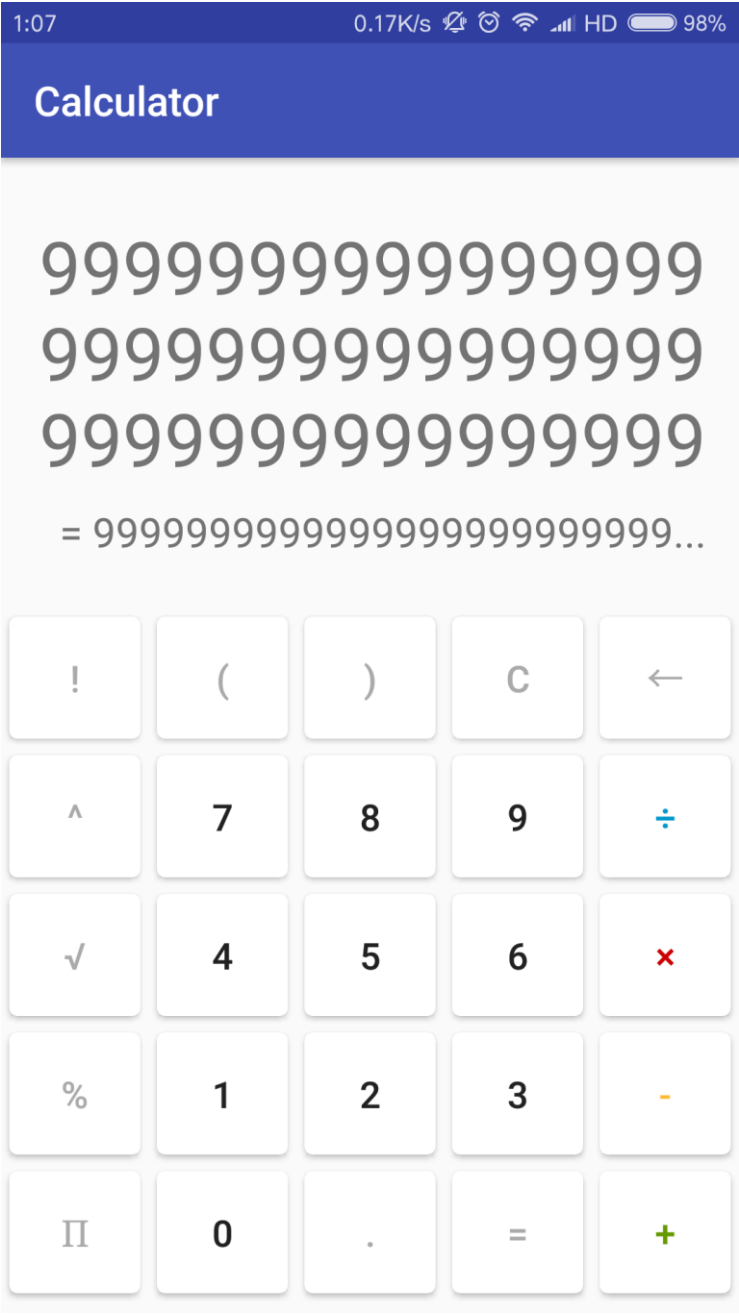


图 3-1 App 遇到用户输入超长算式的显示情况

3.1.2 带有括号、取模、阶乘等多种高级运算符的指令键盘

得益于开发者优秀的算法功底，此计算器能够执行多项高级运算，以较高的性能处理复杂的表达式。

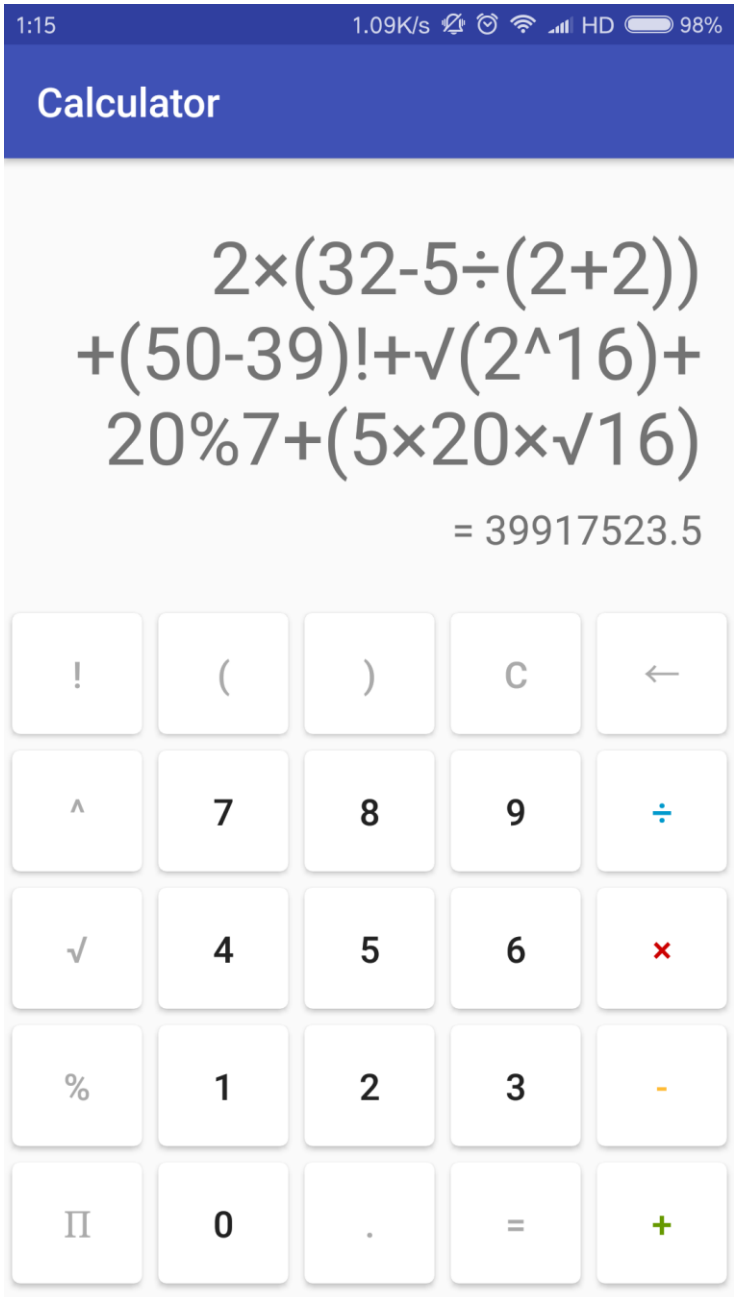


图 3-2 App 对用户输入的高级复杂算式的计算情况

3.1.3 令人称赞的高精度计算

由于 app 使用 `java.math.BigDecimal` 大数类提供的 API 进行计算，除了开方操作对精度有损失之外，所有的运算都将保持小数点后无穷位数的精度。

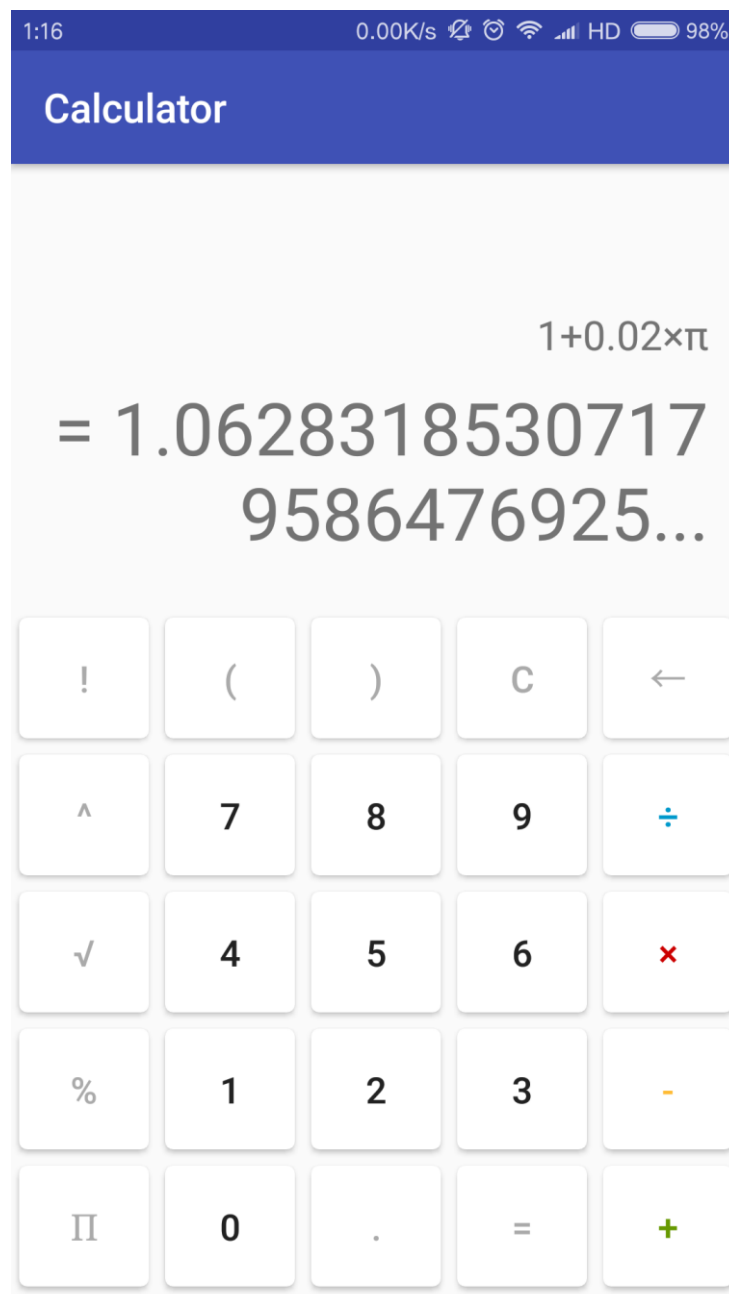


图 3-3 高精度运算

3.1.4 完善的异常处理机制和友好的用户提示信息

开发者本着认真务实的态度，对于用户输入算式的所有异常情况均进行了详尽的分析和处理，使得计算器在出错时给出良好的反馈信息，有效提高了 app 的用户体验。



图 3-4 用户输入非法表达式的情况



图 3-5 用户输入错误的数字格式的情况



图 3-6 用户输入除数为零的情况



图 3-7 用户欲求大数的阶乘导致溢出的情况



图 3-8 用户尝试对非整数求阶乘的情况



图 3-9 用户尝试对负数求阶乘的情况

3.2 卓越的用户体验

3.2.1 契合谷歌 Material Design

开发者对于界面设计非常认真，按照 Material Design 的设计思想，对界面上的按钮的模型重新设计，进行了仔细的打磨，选择了恰到好处的配色方案和按钮的摆放位置。

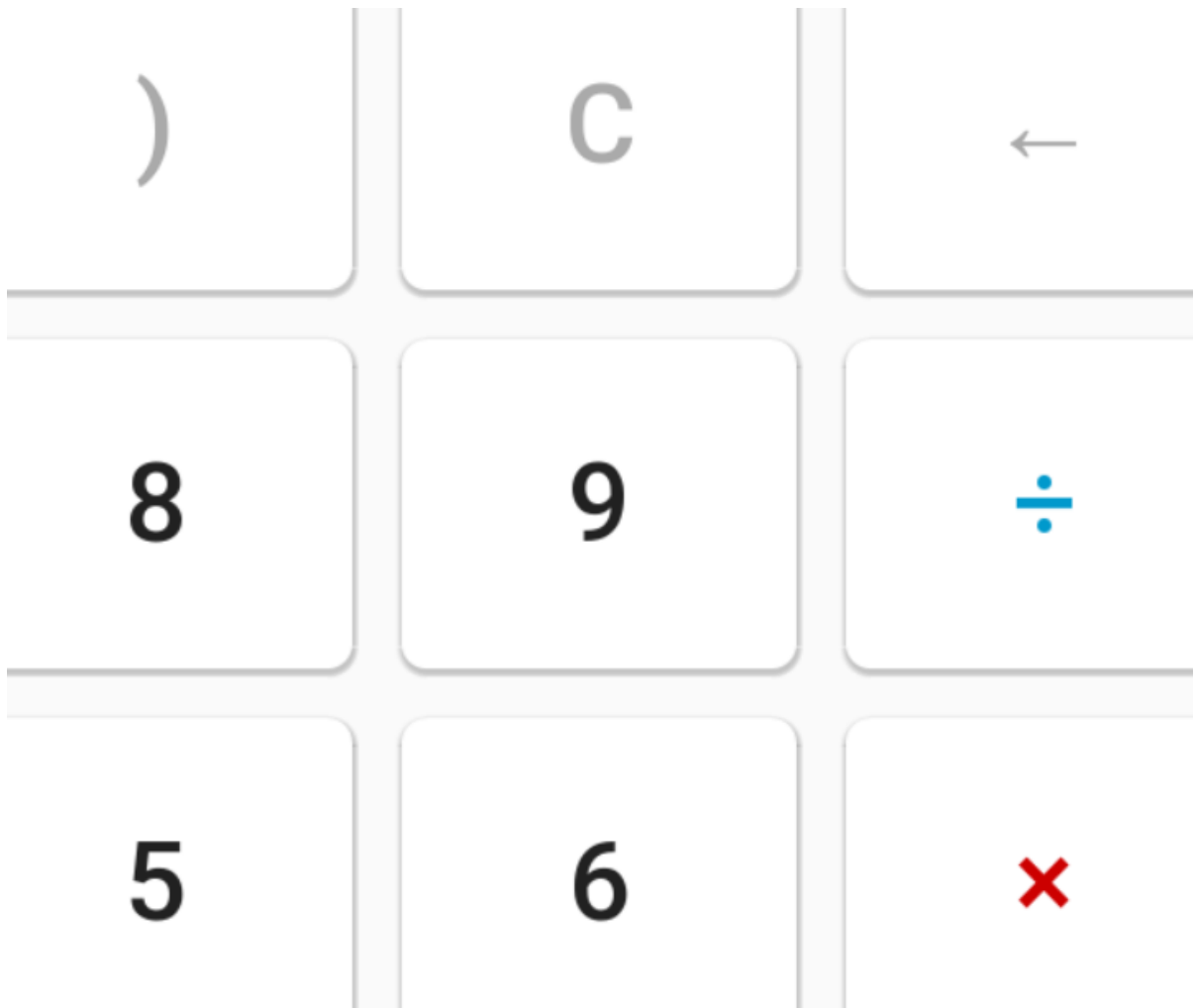


图 3-10 App 按钮模型细节（局部）

3.2.2 细节决定成败的优秀设计

开发者追求完美。

每一种独立的元素颜色，都是开发者对 RGB 三色进行一一穷举后得出的；

每一个按钮的摆放位置，都是开发者进行上百次排列组合尝试后得出的；

每一个界面元素的间隔，都是开发者不胜其烦地进行像素级调试后得出的。

在界面底部，特别留下了 1%的空余位置，是为了避免用户对手机底部菜单栏的误触。

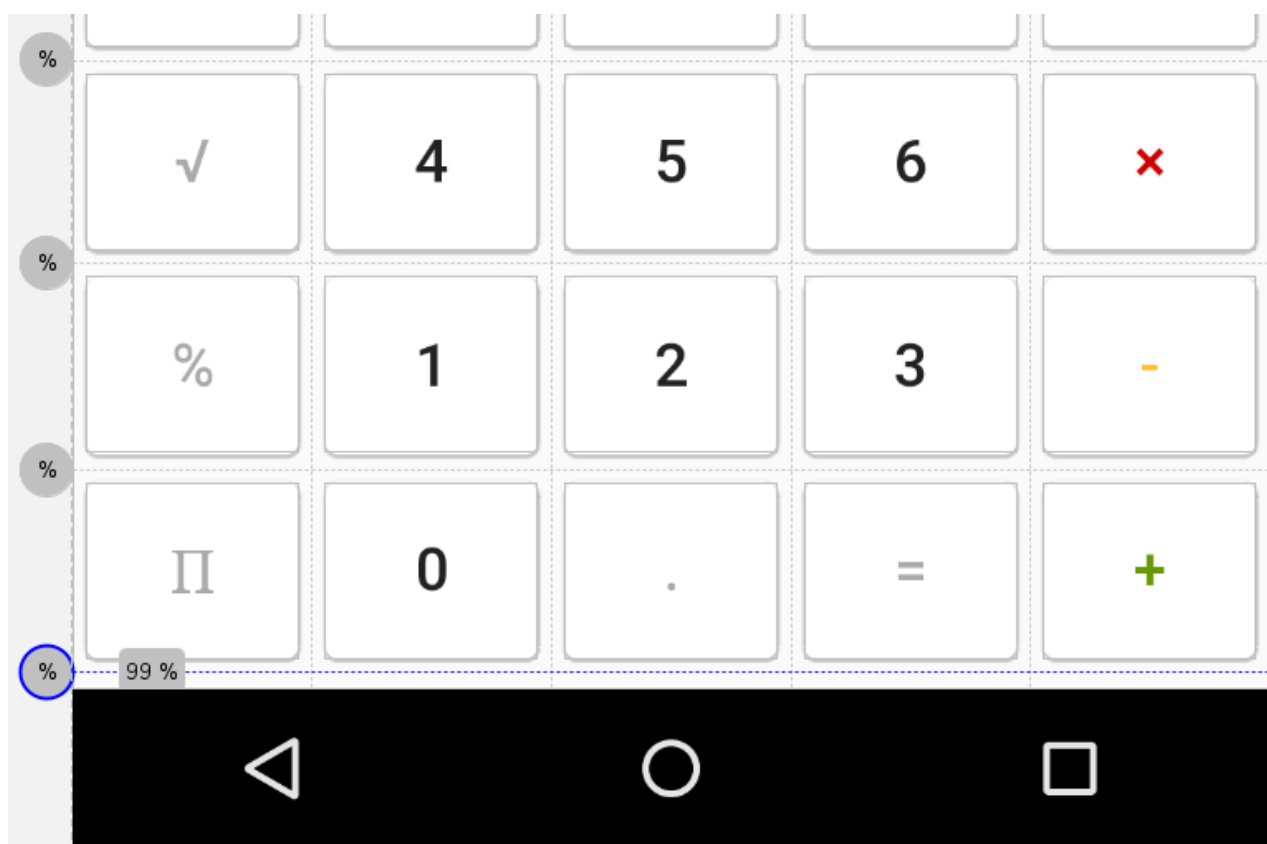


图 3-11 99%的定位参考线，100%的专业

3.2.3 可伸缩高弹性的约束布局的妙用

在本次作业中，开发者对界面布局选用了谷歌新标准中建议的约束布局 (ConstraintLayout)，这种布局能够有效减少 UI 设计中的布局嵌套层次，提高性能和开发效率。结合百分比参考线，这种布局能够适应任意的屏幕比例和大小，并且不像 GridLayout 和 TableLayout 那样笨重，能够轻易地修改按钮的位置。

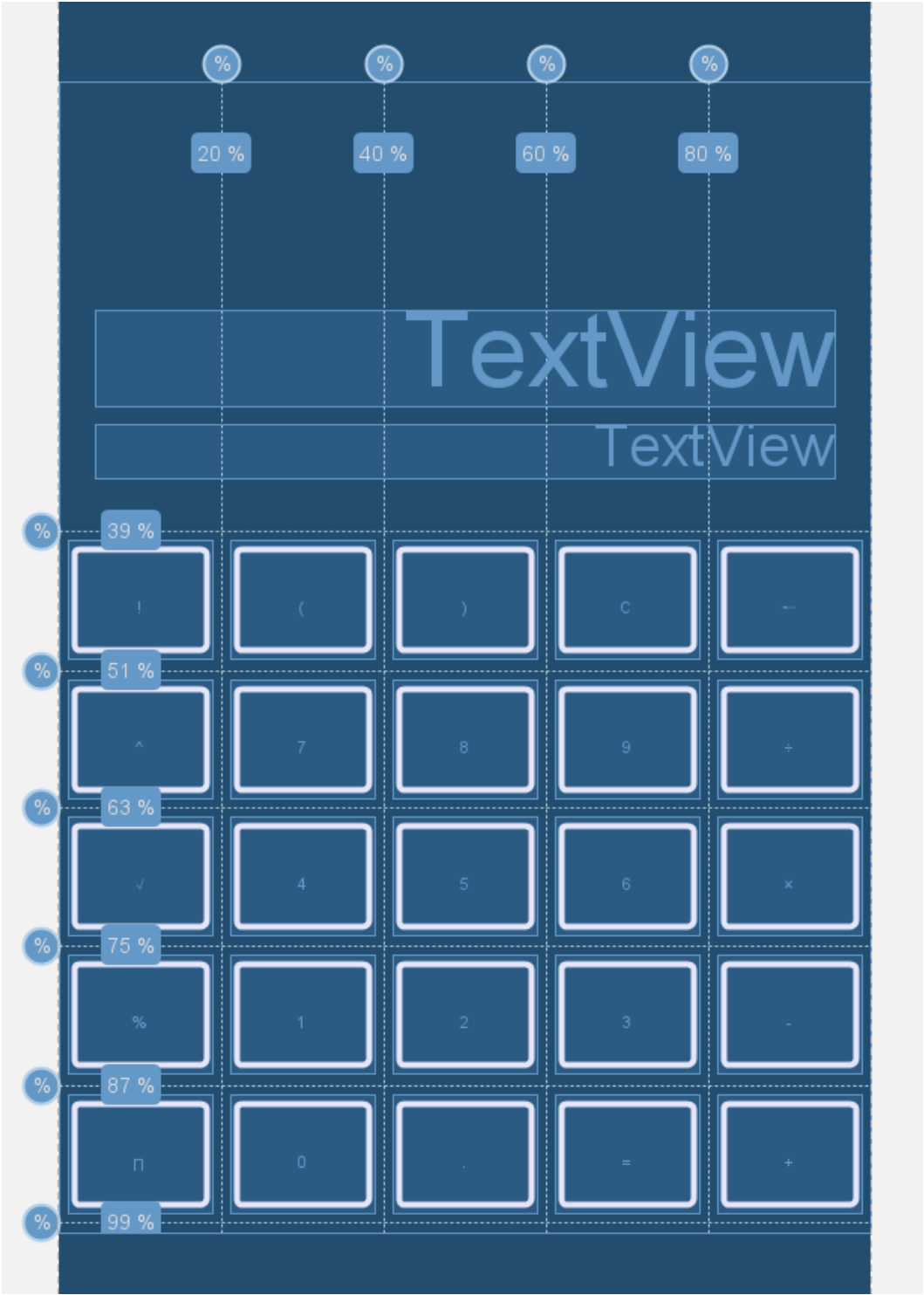


图 3-12 ConstraintLayout 参考线示意图

3.3 即时更新机制

本项目的 app 绑定了永久性下载链接，可在任意时刻进行更新，对用户而言透明，用户无需知道也无法知道开发者对 app 进行了更新，能够确保更新高可用、安全稳定地进行。感谢阿里云对象存储服务（OSS）提供技术支持。

永久下载链接：<http://envoy-public.oss-cn-shenzhen.aliyuncs.com/runnable/calculator.apk>

四、代码设计

由于代码量较大，下面选择几项重要的部分展示。如有兴趣，请在 Android Studio 中 clone 开发者的 Github 代码仓库，打开项目文件进行详细查阅。

```
@SuppressWarnings("SetTextI18n")
private void invalidateResult(boolean isInEqualView) {
    String expr = inputTextView.getText().toString();
    if (expr.equals("0")) return;
    try {
        String result = ExpressionUtils.compute(expr).toPlainString();
        if (result.length() > 25) {
            result = result.substring(0, 25) + "...";
        }
        resultTextView.setText("= " + result);
    } catch (EmptyExpressionException | IllegalExpressionException
            | InvalidComputationException e) {
        if (isInEqualView) {
            resultTextView.setText("非法表达式");
        }
    } catch (NumberFormatException e) {
        resultTextView.setText("数字格式错误");
    } catch (ArithmeticException e) {
        resultTextView.setText("不能除以零");
    } catch (CalculationOverflowException e) {
        resultTextView.setText("溢出");
    } catch (NonIntegerFactorialException e) {
        resultTextView.setText("不能对非整数求阶乘");
    } catch (NegativeFactorialException e) {
        resultTextView.setText("不能对负数求阶乘");
    }
}
```

图 4-1 用户输入任意数字或运算符时进行的表达式计算和异常处理

```

private static BigDecimal _compute(String postExpression)
    throws InvalidComputationException, ArithmeticException, NumberFormatException,
    CalculationOverflowException, NonIntegerFactorialException, NegativeFactorialException {
    Stack<BigDecimal> numberStack = new Stack<>(); // store all numbers and results

    String[] units = postExpression.split(" ");

    for (String currentHandlingString : units) { // loop until there is nothing in string stream
        char currentHandlingChar = currentHandlingString.charAt(0);
        // if currentHandlingString is operator
        if (currentHandlingString.length() == 1 && isOperator(currentHandlingChar)) {
            if (numberStack.size() < 2) { // no enough numbers in stack, failed
                throw new InvalidComputationException();
            }
            BigDecimal operand1 = numberStack.pop();
            BigDecimal operand2 = numberStack.pop();
            switch (currentHandlingChar) { // compute and push
                case '+': numberStack.push(operand2.add(operand1)); break;
                case '-': numberStack.push(operand2.subtract(operand1)); break;
                case '*': numberStack.push(operand2.multiply(operand1)); break;
                case '/': numberStack.push(operand2.divide(operand1, 50, RoundingMode.HALF_EVEN)); break;
                case '%': numberStack.push(operand2.divideAndRemainder(operand1)[1]); break;
                case '^': {
                    double operand1Double = operand1.doubleValue();
                    double operand2Double = operand2.doubleValue();
                    numberStack.push(new BigDecimal(Math.pow(operand2Double, operand1Double)));
                    break;
                }
                case '$': {
                    double operand1Double = operand1.doubleValue();
                    numberStack.push(new BigDecimal(Math.sqrt(operand1Double)));
                    break;
                }
                case '!': {
                    int operand2Int = operand2.intValue();
                    if (operand2.compareTo(BigDecimal.valueOf(operand2Int)) != 0) {
                        throw new NonIntegerFactorialException();
                    }
                    numberStack.push(new BigDecimal(MathUtils.factorial(operand2Int)));
                    break;
                }
            }
        } else { // currentHandlingString is number
            numberStack.push(new BigDecimal(currentHandlingString));
        }
    }

    if (numberStack.size() != 1) { // more than 1 number in stack or no number left, invalid computation
        throw new InvalidComputationException();
    }
    return numberStack.peek(); // the last number in stack is result
}

```

图 4-2 计算过程核心算法

```

package ink.envoy.calculator;

import java.math.BigInteger;

class MathUtils {
    private static final int BOUND = 3201;
    private static BigInteger[] list = new BigInteger[BOUND];
    private static boolean isInitialized = false;

    private static void initialize() {
        list[0] = list[1] = BigInteger.ONE;
        for (int i = 2; i < BOUND; ++i) {
            list[i] = BigInteger.ZERO;
        }
    }

    static BigInteger factorial(int n)
        throws NegativeFactorialException, CalculationOverflowException {
        if (!isInitialized) {
            initialize();
            isInitialized = true;
        }
        if (n < 0) throw new NegativeFactorialException();
        if (n >= BOUND) throw new CalculationOverflowException();
        return _factorial(n);
    }

    private static BigInteger _factorial(int n) {
        if (list[n].compareTo(BigInteger.ZERO) != 0) return list[n];
        return list[n] = _factorial(n - 1).multiply(BigInteger.valueOf(n));
    }
}

class CalculationOverflowException extends Exception {}
class NegativeFactorialException extends Exception {}

```

图 4-3 高精度阶乘核心算法

```

<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- pressed -->
    <item android:state_pressed="true" >
        <shape android:shape="rectangle">
            <corners android:radius="4dp" />
            <solid android:color="#F5F6F8" />
        </shape>
    </item>
    <!-- default -->
    <item>
        <shape
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:shape="rectangle">
            <!-- Corner的属性是设置圆角的半径的-->
            <corners
                android:radius="4dp" />
            <!-- Button里面的文字与Button边界的间隔-->
            <padding
                android:bottom="10dp"
                android:left="10dp"
                android:right="10dp"
                android:top="10dp" />
            <solid android:color="#FFFFFF" />
        </shape>
    </item>
</selector>

```

图 4-4 按钮的模型设计

五、软件操作流程

软件启动后,用户可以通过点击键盘上的按钮来向计算器输入数字或运算符。如果输入错误,可以点击键盘右上角的“←”退格键删除一个字符,或者点击“C”清除键删除全部输入。在用户操作的过程中,屏幕上会实时显示运算结果,并不需要用户点击“=”确认键。当用户输入完毕时,可以点击确认键,软件将会把结果放大显示在屏幕上。这时,得益于软件具有较高的实时性,用户依旧可以修改其输入的算式。当用户使用完毕后,可以点击“C”清除键将计算器复位。

六、难点和解决方案

设计用户在输入算式时，如何对运算结果进行实时更新是本项目最大的难点。经过思考，可以得出以下方案：先分析用户输入的算式是否正确，然后进行预处理，再用逆波兰法将用户输入的中缀表达式转换为后缀表达式，然后将表达式传入计算引擎进行计算。以下是项目中将中缀表达式转换为后缀表达式的代码：

```
// 中缀转后缀核心算法
private static String toPostfix(String expr) {
    expr += "#";
    String result = "";
    Stack<Character> operatorStack = new Stack<>();
    operatorStack.push('#');

    for (int i = 0; i < expr.length(); ) {
        char c = expr.charAt(i);
        if (isDigit(c) || c == '.') { // 如果c是数字，放入结果中
            result += c;
            if (!isDigit(expr.charAt(i + 1)) && expr.charAt(i + 1) != '.') { // 确保结果中的数字在一起显示
                result += " ";
            }
            ++i;
        } else if (isOperator(c)) {
            char topChar = operatorStack.peek();
            if (getIcp(c) > getIsp(topChar)) { // 如果 icp(c) > isp(op)，令c入栈并继续循环
                operatorStack.push(c);
                ++i;
            } else { // 如果 icp(c) <= isp(op)，弹栈
                operatorStack.pop();
                if (getIcp(c) < getIsp(topChar)) { // 如果 icp(c) < isp(op)，令op入栈
                    result += topChar;
                    result += " ";
                } else if (leftBraces.contains(topChar)) { // 循环至c和栈顶字符均为左括号，再继续读入下一字符
                    ++i;
                }
            }
        } else if (c == '#') { // 到达字符串尾部，将所有操作数放入结果中并结束
            if (operatorStack.peek() == '#')
                break;
            result += operatorStack.pop();
            result += " ";
        }
    }
    return result;
}
```

图 6 中缀表达式转为后缀表达式核心算法

该算法完美解决了如何对运算结果进行实时更新的问题。

七、不足之处和今后的设想

本次项目历时较短，未能投入足够多的时间设计和开发 app，有着些许不足之处，计算器功能有限，未能开发出更多的附加功能，如夜间模式、汇率转换、税率计算、单位转换等。正如报告书中的代码所示，最大的不足之处在于没有实现国际化通用，忽略了 I18N 的重要性，将 app 内的一些字符串写成 hard code，无法被翻译成其他语言。开发者将来设计的 app 将遵守 I18N 的规范，为国际化做好充足的准备。

