一、使用方法

在下拉选项框中选择要处理的函数，在文本框中输入函数，回车保存函数信息，点击OK键进行函数绘制，函数可以是y=f（x）形式也可以直接是f(x)形式，Enlarge实现放大，Reduce实现缩小或者用鼠标滑轮的滚动实现放大或缩小，向上滑动实现放大，向下滑动实现缩小，点击Clear实现清空函数图像重置缩放比例。

除了在下拉选项框中选择要处理的函数，也可以通过上下键实现选择函数，一次可以最多绘制三个有效函数，可以根据弹出的对话框了解到函数是否有效，若要将某个函数清除则在对应的选项框将文本域清空或输入空格处理即可。故可以一次输入两个有效函数实现两个函数的绘制。

点击OK在函数绘制的同时实现同步保存绘图文件到项目文件下的images文件夹中，工具日期和时间进行文件区分

注：

1,函数不区分乘号是否书写，xx和x^2是一个函数，xxx和x^3是一个函数。

2,此外不区分y=f(x)形式和f(x)形式，但是对于三角函数，sinx和sin(x)是一个函数，但是sinx+1和sin(x)+1是两个函数，sinx+1对应着sin(x+1),若严格书写函数表达式那么将得到更好的体验

二、代码原理：

1. Java绘图AWT，SWING
2. 插件主窗口JFrame jf
3. 设置布局管理器模式为BorderLayout()
4. 创建下拉选择框functionChooser，三个选项function1，function2和function3，分别用于区分三个函数的输入保存信息
5. 创建容器jPanel1，向其中添加两个文本域(tips(不可更改)，functionField(函数输入域))，在两个文本域中间添加下拉选项框functionChooser对输入的文本进行解析绘制，和一个okButton按键，为okButton添加一个监控器，当按下okButton时对函数解析绘制
6. 再将jPanel1添加到jf中（SOUTH位置处）
7. 创建容器jPanel2，向其中添加一个画布对象mc(MyCanvas继承自画布Canvas),将Panel2置于jf的CENTER位置处
8. 创建容器jPanel3，向其中添加按键biggerButton，按键smallerButton和按键clearButton，对这个三个按键创建一个方法实现添加监控器，分别实现放大，缩小和清空，实现原理时通过一个缩放比例参数scale，再其次像jPanel3添加文本域scaleShow，实时展示用户调节后的scale
9. 将jPanel3添加到jf中，再分别设置jf的长宽度，可见，默认关闭方式，置于windows窗口中间位置

functionChooser的检测原理：

添加一个监控器，每次选择的时候进行判断，将文本域的内容呈现所选的函数

functionField的检测原理：

1,为functionField添加一个回车按键监控器

2,回车之后调用saveFunction方法，进行函数解析，若函数为空则弹出函数为空警告对话框， 若函数式不为空，则进行函数式子有效性的检验，无效则置弹出函数无效对话框，文本域为空，canDraw置为false，有效则保存函数对对应选项，canDraw置为true

3.为functionField添加上下键的监控器，实现快速切换对应的函数选项，方便人机交互（实现原理为更改选项的下标，更改选项后将文本域置为更改之后的选项

okButtion的监控器原理：

点击ok之后，进行两种判断，若三个函数式都为空，则弹出三个函数式都为空警告对话框，否则 调用mc的repaint()方法实现绘制，点击ok的同时调用FunctionSave对象fSave中的saveImage方法，传递mc对象，传递三个函数，实现函数绘制与保存

鼠标滑动放缩的实现原理：

为jf添加一个鼠标滚轮监控器：鼠标滚轮上滑时增大myScale（缩放因子），鼠标滚轮下滑时减小myScale，但是控制myScale的大小位于0.25到0.5之间防止过度缩放，当过度缩放时调用sd（showDialog类的对象）的showScaleMessageDialog方法弹出警告对话框

两个按钮的放缩及清空的实现原理：

biggerButton(Enlarge)按键和smallerButton(Reduce)按键(对外展示Enlarge和Reduce这两个吃灰更加高级一些) 调用方法addButtonListener为这两个按键添加监控器，具体实现和鼠标监控器类似，再用addButtonListener为clearButton添加监控器，对于clearButton的监控器，只需要将myScale重置为1.0再将canDraw设置为false，清空函数文本域即可

核心代码：

MyCanvas的自定义paint，获取画笔，再重置原点坐标，先绘制背景方格再为画布设置缩放比例，然后就是绘制坐标轴：核心就是绘制刻度，绘制刻度是每通过for循环每20个像素点设置为一个刻度单位1，随后绘制原点，再通过for循环来绘制函数的每个点，用对象fs调用方法calculateFunction来计算出每个x对应的y值，再绘制小圆点，用MAXSIZE（5000）来代替MYWIDTH/MYHEIGHT来绘制是为了满足缩放要求

关于函数的绘制：

若canDraw为true则进行函数绘制

为了使y的绘制能够与刻度相对应，并且使原点绘制清晰，每次计算时对x/100相当于刻度上的-50到50，举个例子：此时x为20，20/100为0.2，此时y为0.2，绘制图像时x/5为4，y\*100/5为4，就是在4，4处绘制了一个点，代码核心就是实现与图像匹配，20个像素点为单位刻度1，若1个像素点为单位刻度，那么图像显示就靠近x轴(如sinx)

需要明确区分实际x和y的值和图像上x和y的值

关于函数绘制保存的原理：  
主要是利用JavaIO流中ImageIO和FileWrite实现文件的读写

创建BufferedImage的对象image，从image中获取画笔g2，再同等的绘制MyCanvas中的paint即可，再将image保存为jpg文件

同样，利用文件FileWirte类中的write实现将字符串写入到functions.txt中

将上述的.jpg文件和.txt文件保存在多级文件夹image//date//time中实现即使多次绘制的文件的位置也非常清晰

三、四个类的解释以及外部包资源

FunctionDraw类：主类，实现主要功能，最主要的方法为其中的init()方法，将设置的变量初始化，实现各种监控器功能以及绘图功能

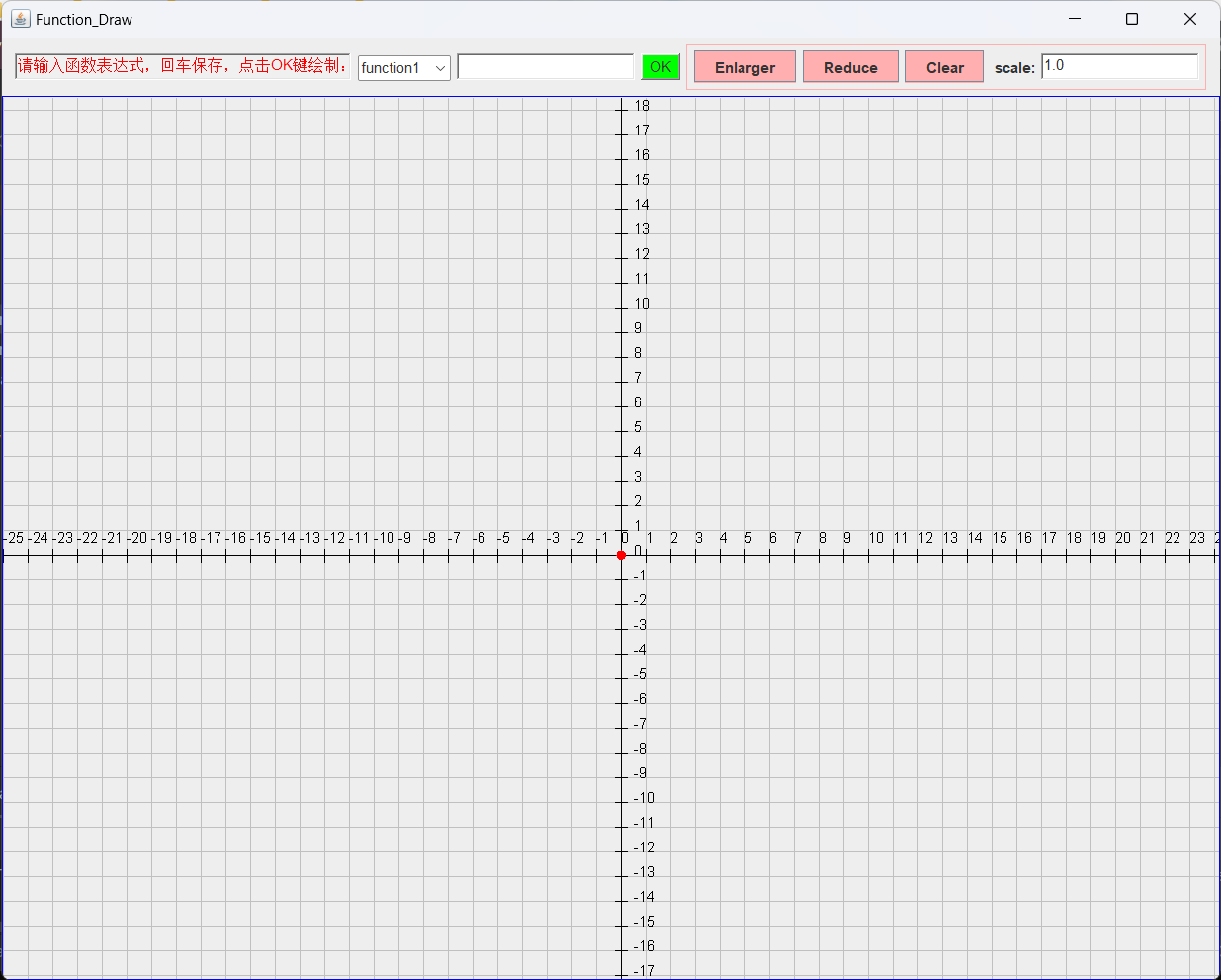
FunctionSolution类：主要用于处理函数，计算函数，其中包含3个方法：getFunction方法用来解析函数文本域，用来获取函数解析式，解析y=f(x)和x两种格式。checkFunction方法用来检查函数是否正确，调用了exp4j包的Expression类来解析表达式。calculateFunction方法用来计算函数值，将Expression里面的x变量值更改实现计算功能

ShowDialog类：主要包含五个方法，showEnterWarningDialog方法，，showEmptyWarningDialog方法，showScaleMessageDialog方法，showThreeFunctionsEmpty方法，showFunctionSavedDialog方法，分别实现弹出 函数输入错误警告对话框，函数输入为空警告对话框，缩放过度警告对话框，三个函数均为空警告对话框，函数已保存提示对话框

FunctionSave类：主要包含saveImage方法，实现将绘制的函数保存到images对应的文件中，实现原理见代码原理的最后一部分

四、截图部分

程序原始界面：



对应代码：

class MyCanvas extends Canvas {

@Override

public void paint(Graphics g) {

showScale.setText(Double.toString(myScale));

//获取画笔

Graphics2D g2 = (Graphics2D) g;

g2.translate(MYWIDTH / 2, MYHEIGHT / 2 - 30); //主要是考虑到上面有个panel模块

//将中心坐标轴移动到MYWIDTH/2 MYHEIGHT/2-30的位置处

g2.setColor(Color.LIGHT\_GRAY);

for (int i = -MAXSIZE; i <= MAXSIZE; i += 20) {

g2.drawLine(i, -MAXSIZE / 2, i, MAXSIZE / 2); //-25到25左右

}

for (int i = -MAXSIZE / 2; i <= MAXSIZE / 2; i += 20) {

g2.drawLine(-MAXSIZE / 2, i, MAXSIZE / 2, i); //-19到19左右

}

//绘制小方格，不需要改动

g2.scale(myScale, myScale);

//设置缩放

//绘制x轴和y轴

g2.setColor(Color.BLACK);

g2.drawLine(-MAXSIZE / 2, 0, MAXSIZE / 2, 0);

g2.drawLine(0, -MAXSIZE / 2, 0, MAXSIZE / 2);

//绘制刻度

for (int i = -MAXSIZE / 2; i <= MAXSIZE / 2; i += 20) {

g2.drawLine(i, -5, i, 5);

g2.drawString(Integer.toString(i / 20), i, -10);

//每20个像素点为1个刻度单位

}

for (int i = -MAXSIZE / 2; i <= MAXSIZE / 2; i += 20) {

g2.drawLine(-5, i, 5, i);

g2.drawString(Integer.toString(-i / 20), 10, i);

}

//绘制原点

g2.setColor(Color.RED);

g2.fillArc(-4, -4, 8, 8, 0, 360);

//用宽8长8的实心小圆点覆盖0 0 坐标实现原点的绘制

g2.setColor(Color.RED);

if (canDraw1) {

for (int x = -MAXSIZE \* 2; x <= MAXSIZE \* 2; x++) {

double y = fs.calculateFunction(function1, (double) x / 100); //该函数这里传入的是一个double的值

if (y >= -MAXSIZE / 2 && y <= MAXSIZE / 2) {

//System.out.println("here x/5="+x/5+" y\*20="+y\*20);

g2.fillOval(x / 5, -(int) (y \* 100) / 5, 2, 2);

//区分x,y的实际值和图像的值

//y的计算处/100之后又\*20相当于x/5，保持了一样的缩放比例

//每拖动一次计算一次造成计算冗余

}

}

}

//canDraw1参数为true是绘制函数，处理函数的缩放，实现于坐标轴相对应

if (canDraw2) {

for (int x = -MAXSIZE \* 2; x <= MAXSIZE \* 2; x++) {

double y = fs.calculateFunction(function2, (double) x / 100);

if (y >= -MAXSIZE / 2 && y <= MAXSIZE / 2) {

g2.fillOval(x / 5, -(int) (y \* 100) / 5, 2, 2);

}

}

}

if (canDraw3) {

for (int x = -MAXSIZE \* 2; x <= MAXSIZE \* 2; x++) {

double y = fs.calculateFunction(function3, (double) x / 100);

if (y >= -MAXSIZE / 2 && y <= MAXSIZE / 2) {

g2.fillOval(x / 5, -(int) (y \* 100) / 5, 2, 2);

}

}

}

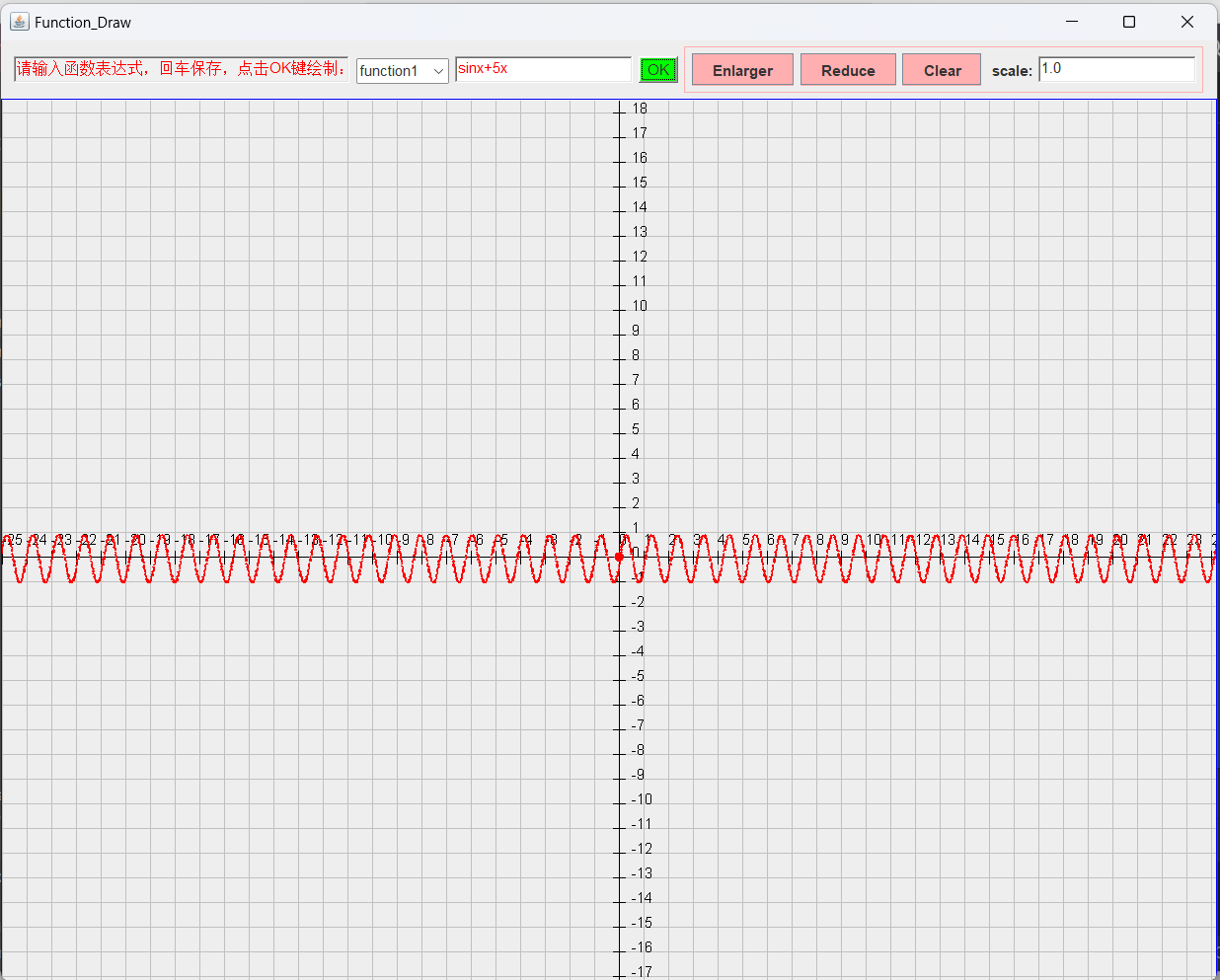
//三个绘制代码互不相干

}

}

//自定义的画布类，实现对画布对象mc的绘制功能

以y=sinx+5x为例，绘制界面：

缩放界面1 ：  


对应代码：  
jf.addMouseWheelListener(new MouseWheelListener() {

@Override

public void mouseWheelMoved(MouseWheelEvent e) {

int notches = e.getWheelRotation();

if (notches < 0) {

if (myScale > 5.0 || myScale < 0.25) {

myScale = 1;

sd.showScaleMessageDialog(jf);

} else myScale \*= 1.1;

} else {

if (myScale > 5.0 || myScale < 0.25) {

myScale = 1;

sd.showScaleMessageDialog(jf);

} else

myScale /= 1.1;

}

mc.repaint();

}

});

//为主窗口jf添加鼠标滚轮监控器，处理放大缩小事务

public void addButtonListener(JButton b) {

b.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (e.getActionCommand().equals("Enlarger")) {

if (myScale > 5.0 || myScale < 0.25) {

myScale = 1;

sd.showScaleMessageDialog(jf);

} else

myScale \*= 1.1;

} else if (e.getActionCommand().equals("Reduce")) {

if (myScale > 5.0 || myScale < 0.25) {

myScale = 1;

sd.showScaleMessageDialog(jf);

} else

myScale /= 1.1;

} else if (e.getActionCommand().equals("Clear")) {

myScale = 1;

functionField.setText("");

function1="";

function2="";

function3="";

canDraw1 = false;

canDraw2 = false;

canDraw3 = false;

}

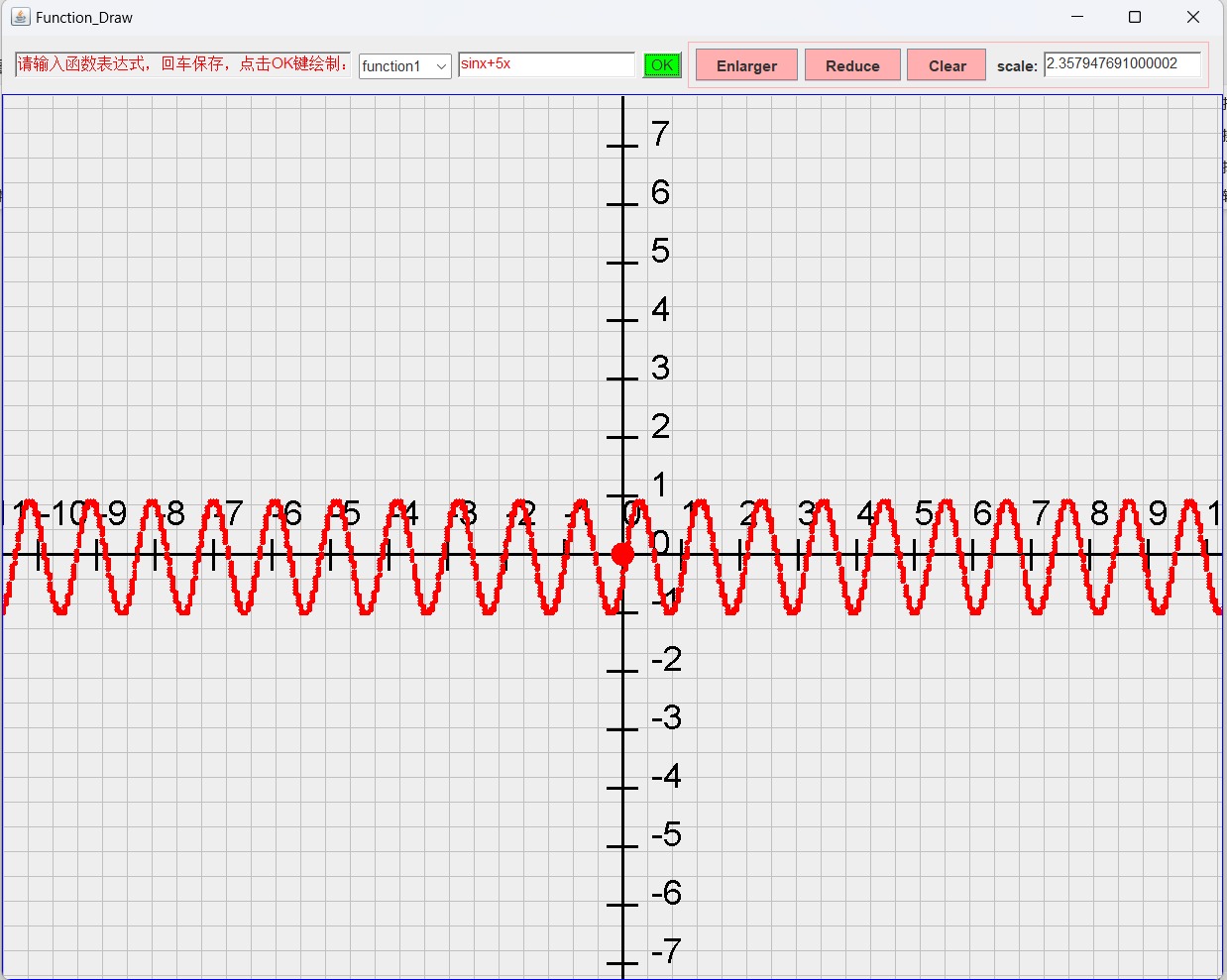
mc.repaint();

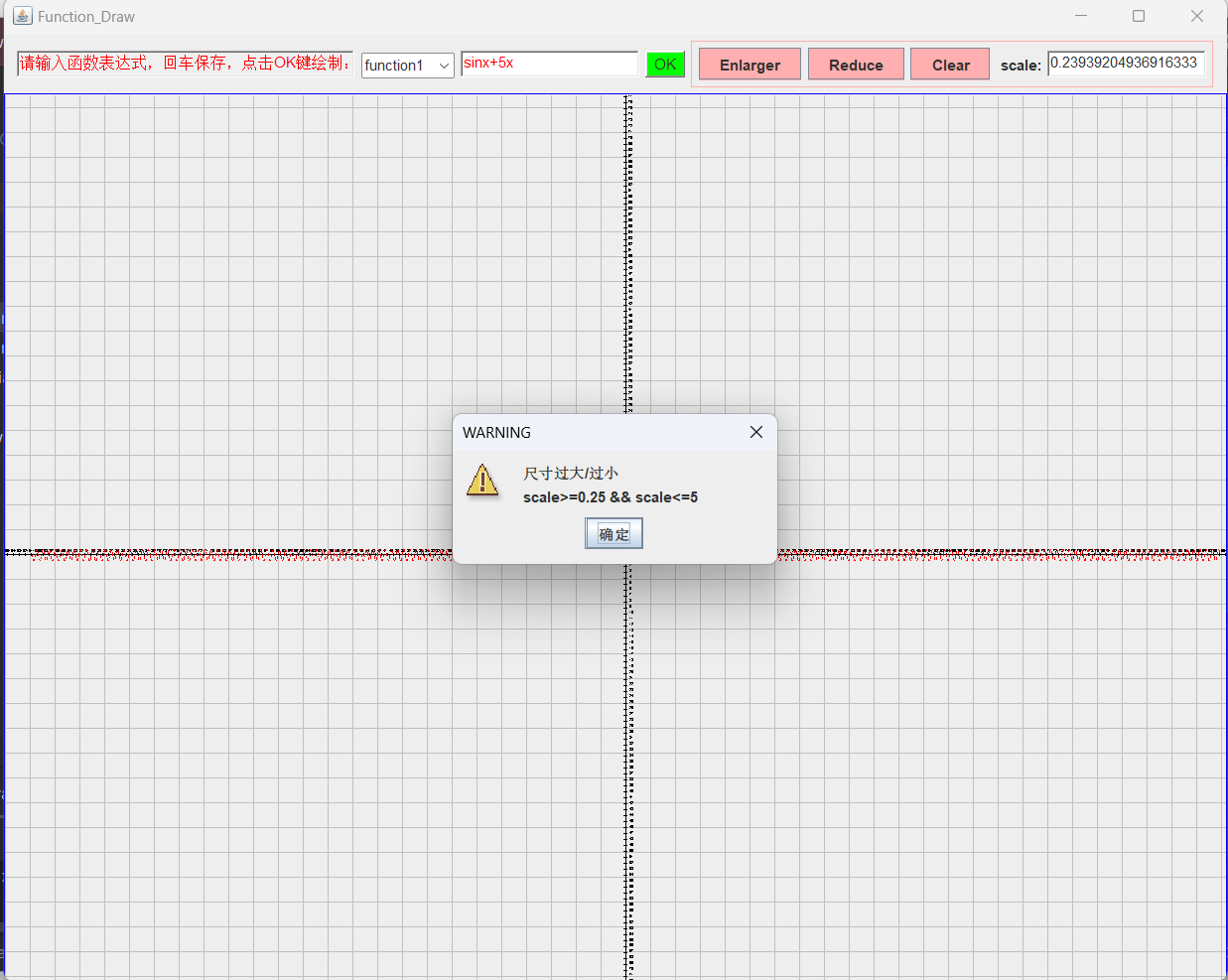
}

});

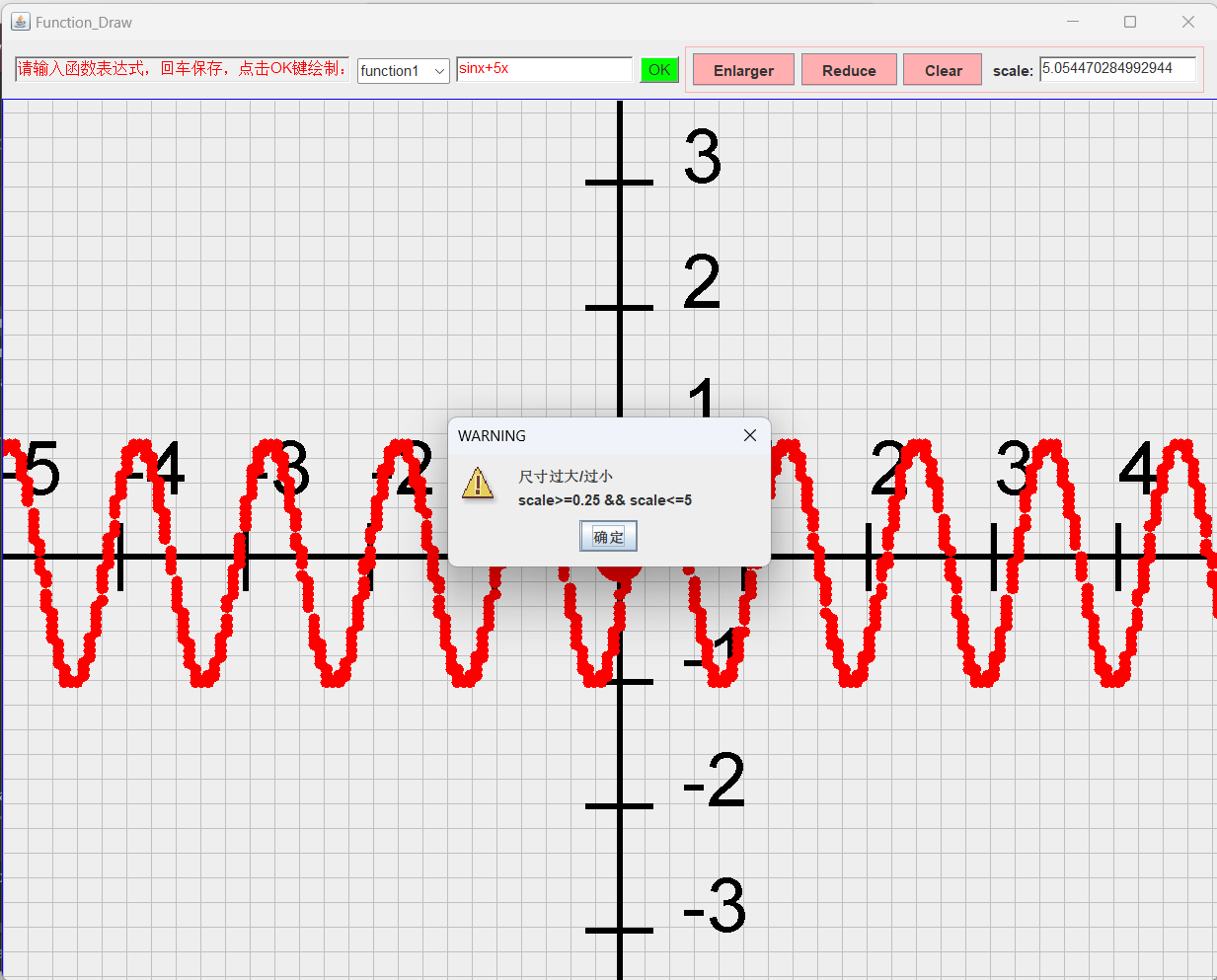
//自定义方法，为放大，缩小清空按钮添加监控事件，用一个方法去处理，减少代码冗余

}

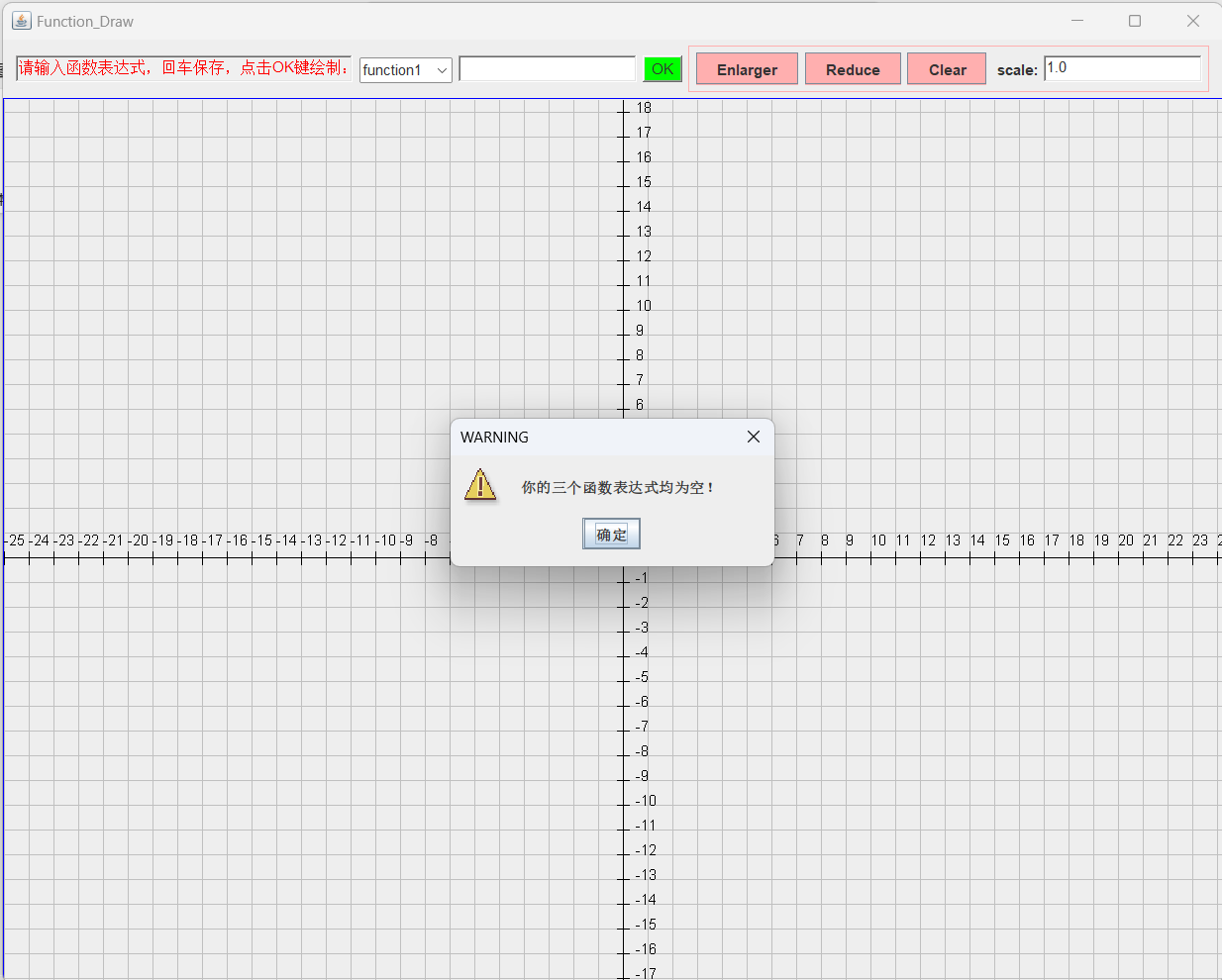
缩放界面2：  


缩放过度警告对话框界面1：  


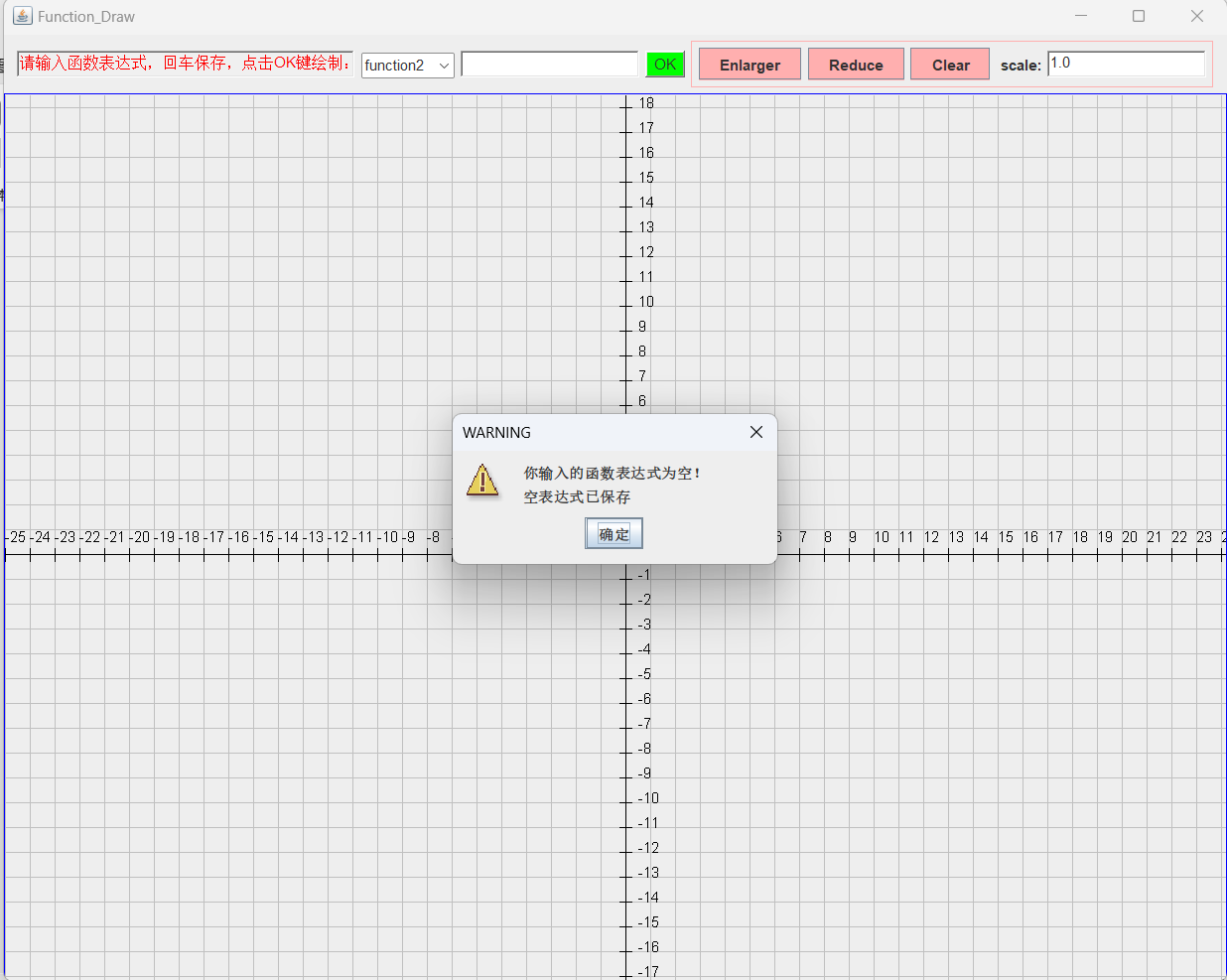
缩放过度警告对话框界面2：

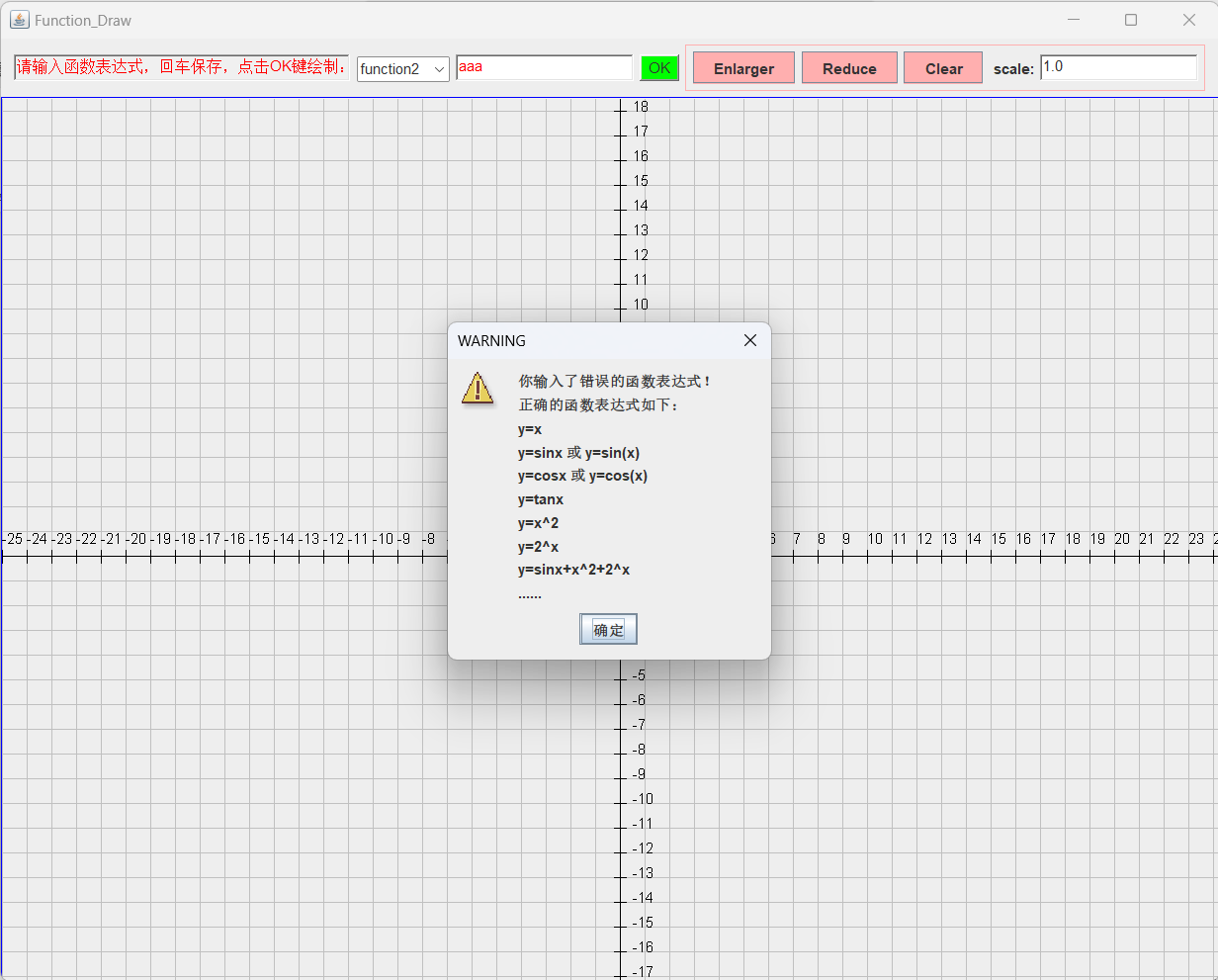


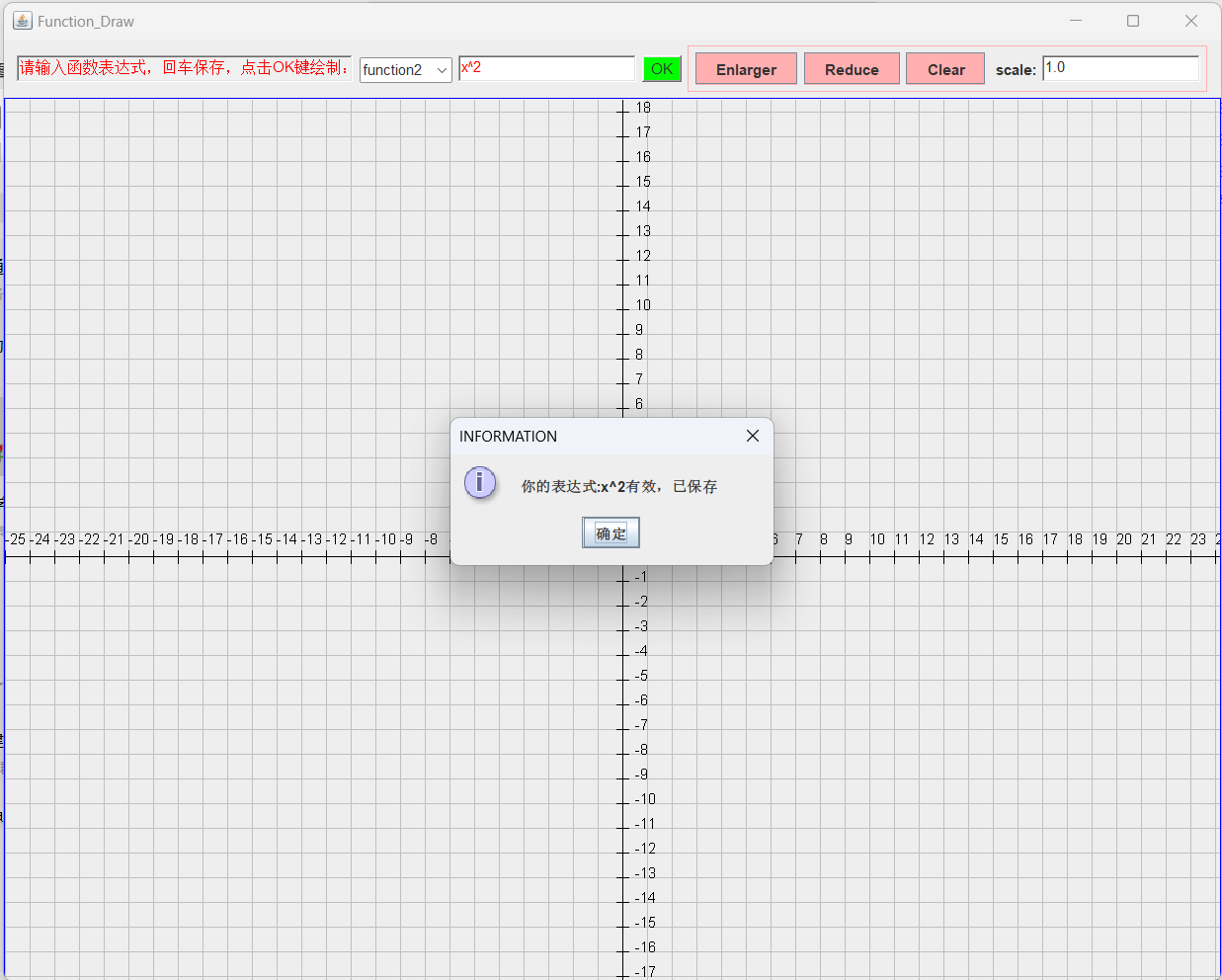
函数表达式为空警告界面1：



函数为空警告2：



函数输入有误警告界面：  


函数有效保存界面：  


对应代码：  
package practice;

import javax.swing.\*;

public class ShowDialog {

public void showEnterWarningDialog(JFrame jf){

String message = "你输入了错误的函数表达式！\n正确的函数表达式如下：\n" +

"y=x\n" +

"y=sinx 或 y=sin(x)\n"+

"y=cosx 或 y=cos(x)\n"+

"y=tanx\n"+

"y=x^2\n" +

"y=2^x\n"+

"y=sinx+x^2+2^x\n"+

"......";

JOptionPane.showMessageDialog(jf, message, "WARNING", JOptionPane.WARNING\_MESSAGE);

}

//处理表达式错误

public void showEmptyWarningDialog(JFrame jf){

JOptionPane.showMessageDialog(jf, "你输入的函数表达式为空！\n"+

"空表达式已保存", "WARNING", JOptionPane.WARNING\_MESSAGE);

}

//处理输入的表达式为空

public void showScaleMessageDialog(JFrame jf){

JOptionPane.showMessageDialog(jf, "尺寸过大/过小\n" +

"scale>=0.25 && scale<=5", "WARNING", JOptionPane.WARNING\_MESSAGE);

}

//处理放缩过度

public void showThreeFunctionsEmpty(JFrame jf){

JOptionPane.showMessageDialog(jf, "你的三个函数表达式均为空！", "WARNING", JOptionPane.WARNING\_MESSAGE);

}

//处理三个表达式均为空的情况

public void showFunctionSavedDialog(JFrame jf,String function){

JOptionPane.showMessageDialog(jf, "你的表达式:"+function+"有效，已保存", "INFORMATION", JOptionPane.INFORMATION\_MESSAGE);

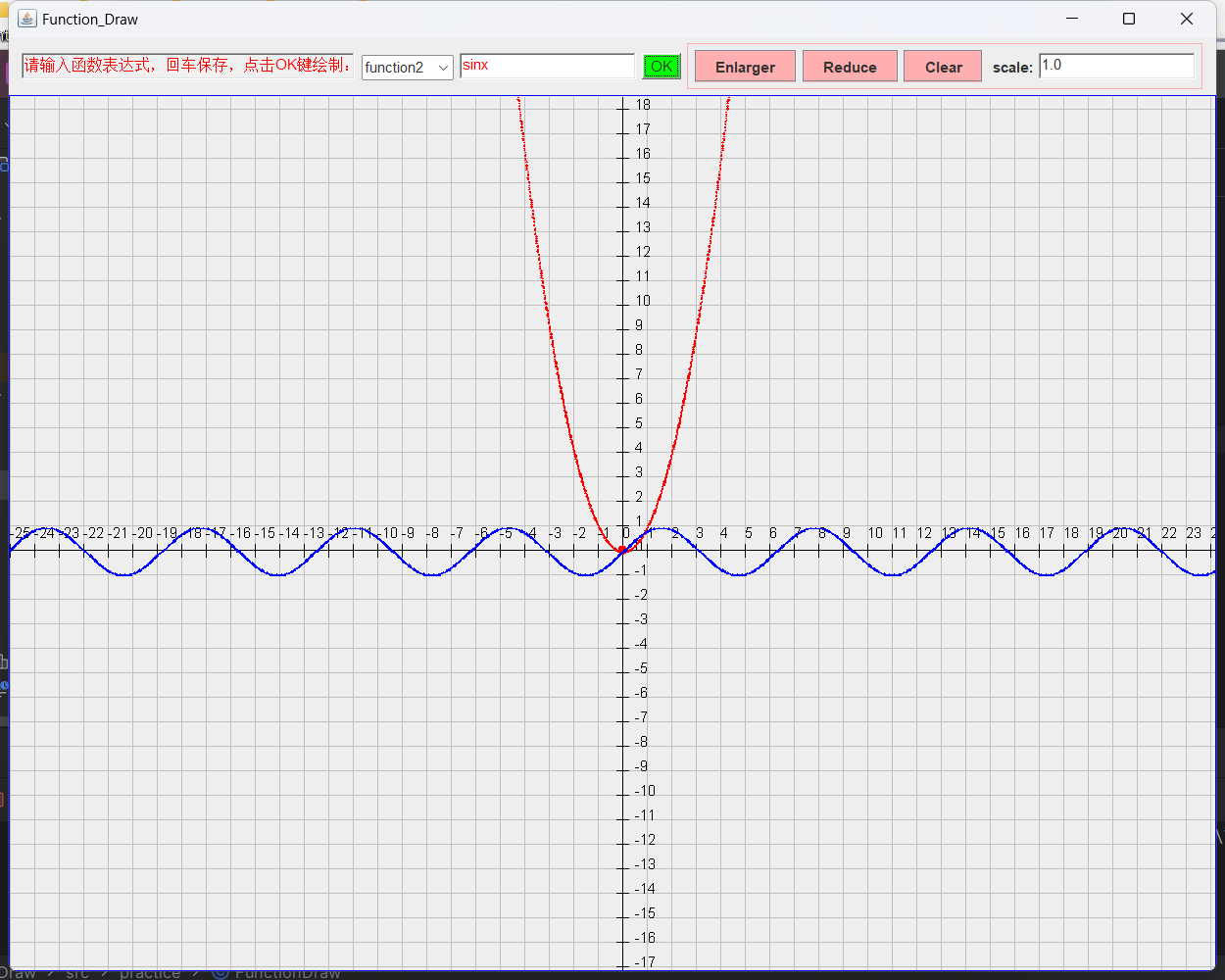
}

//弹出保存信息

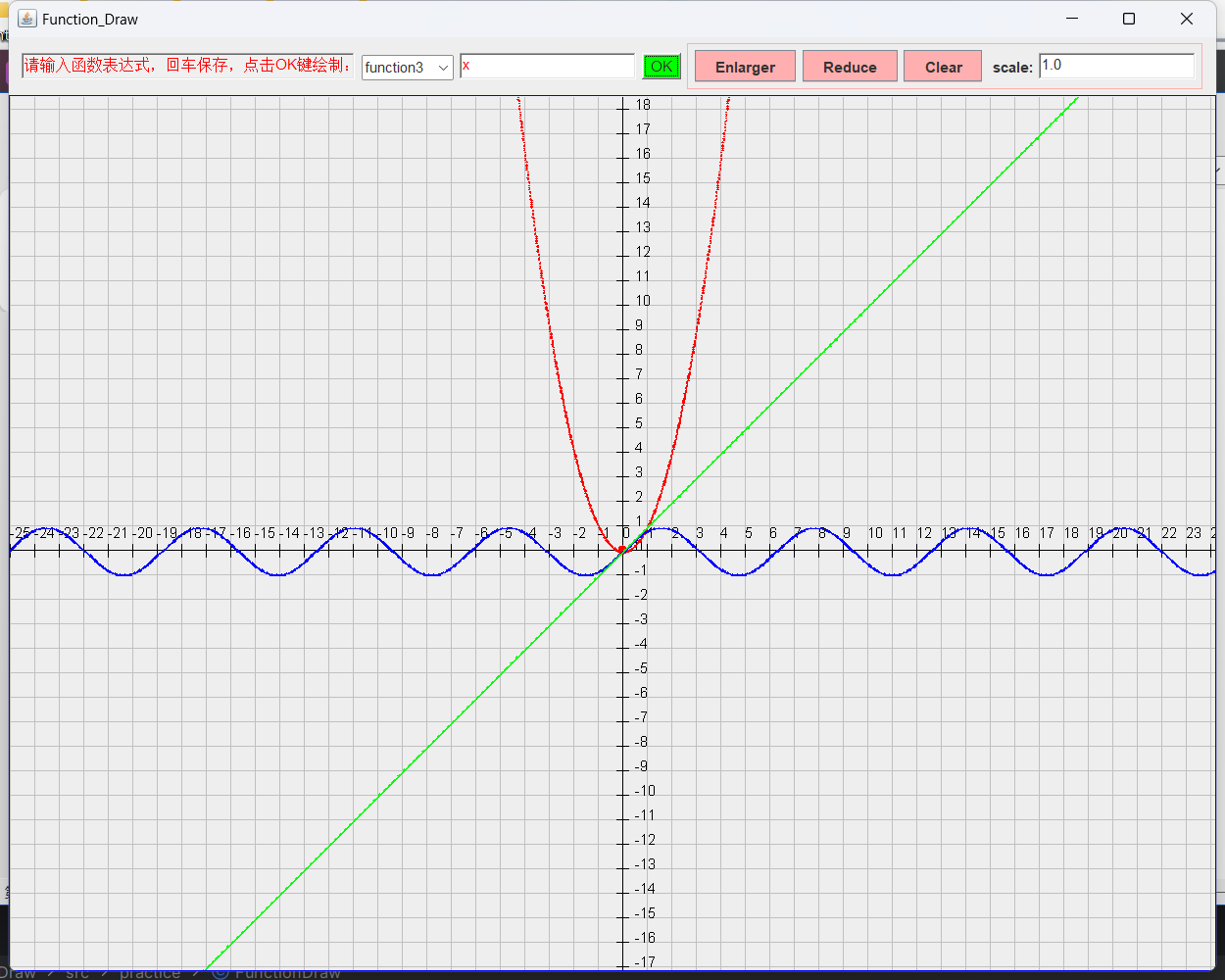
}

//用来展示在绘制函数过程中遇见的各种问题

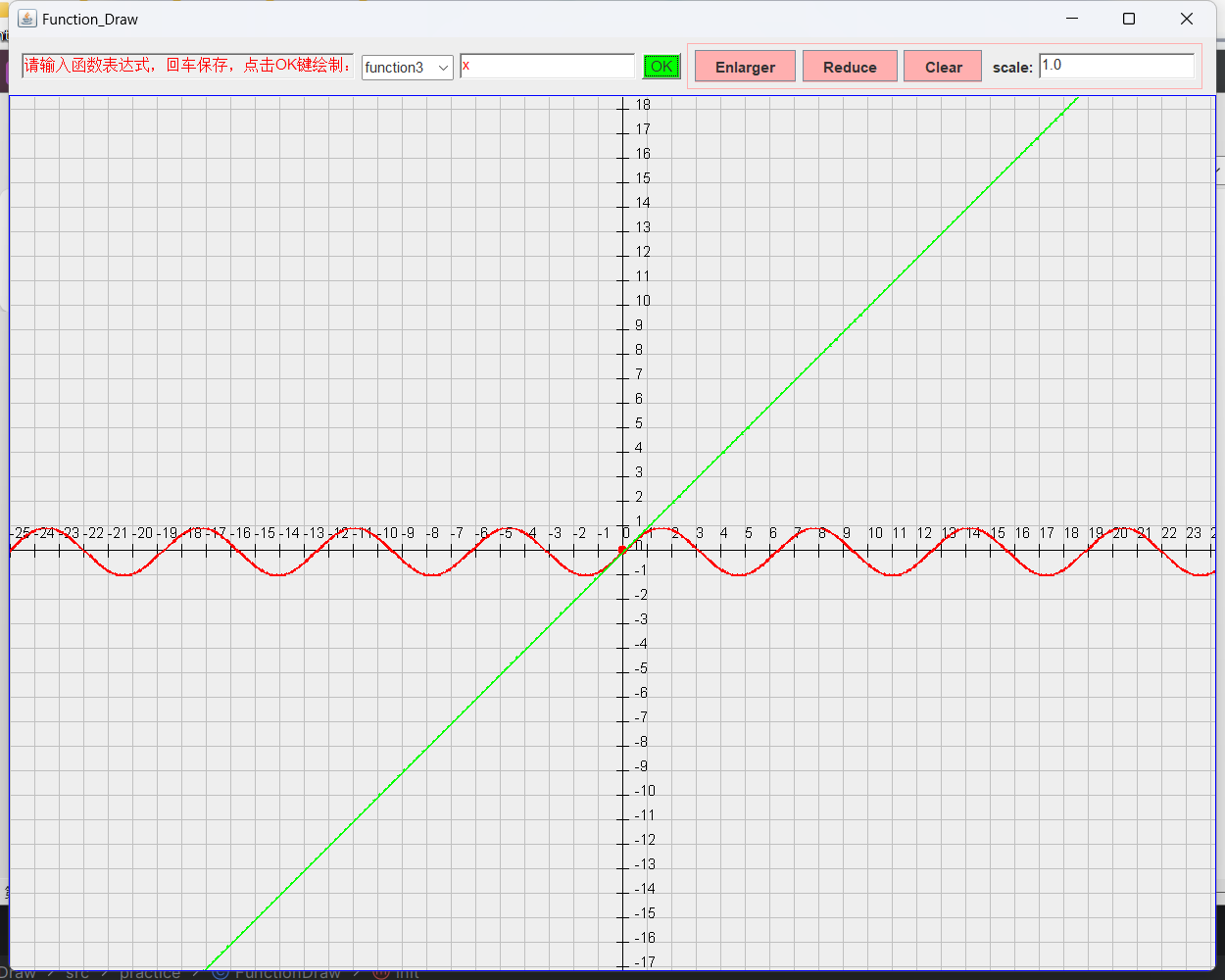
同时绘制两个函数截图：



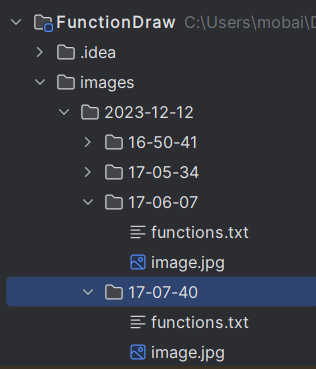
同时绘制三个函数截图：

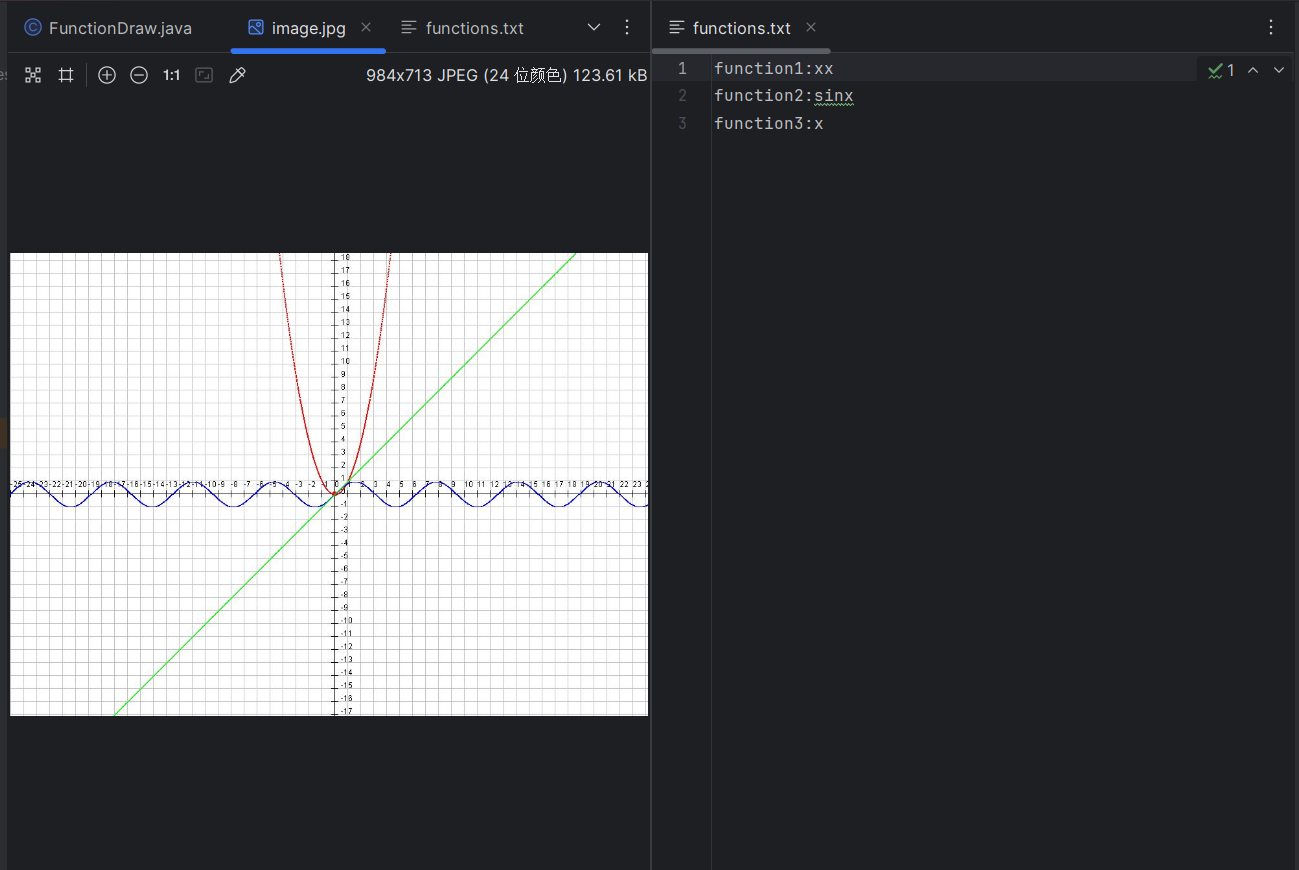


已知function1为sinx，function2为x^2，function3为x

删去中间一个函数 函数绘制截图：  


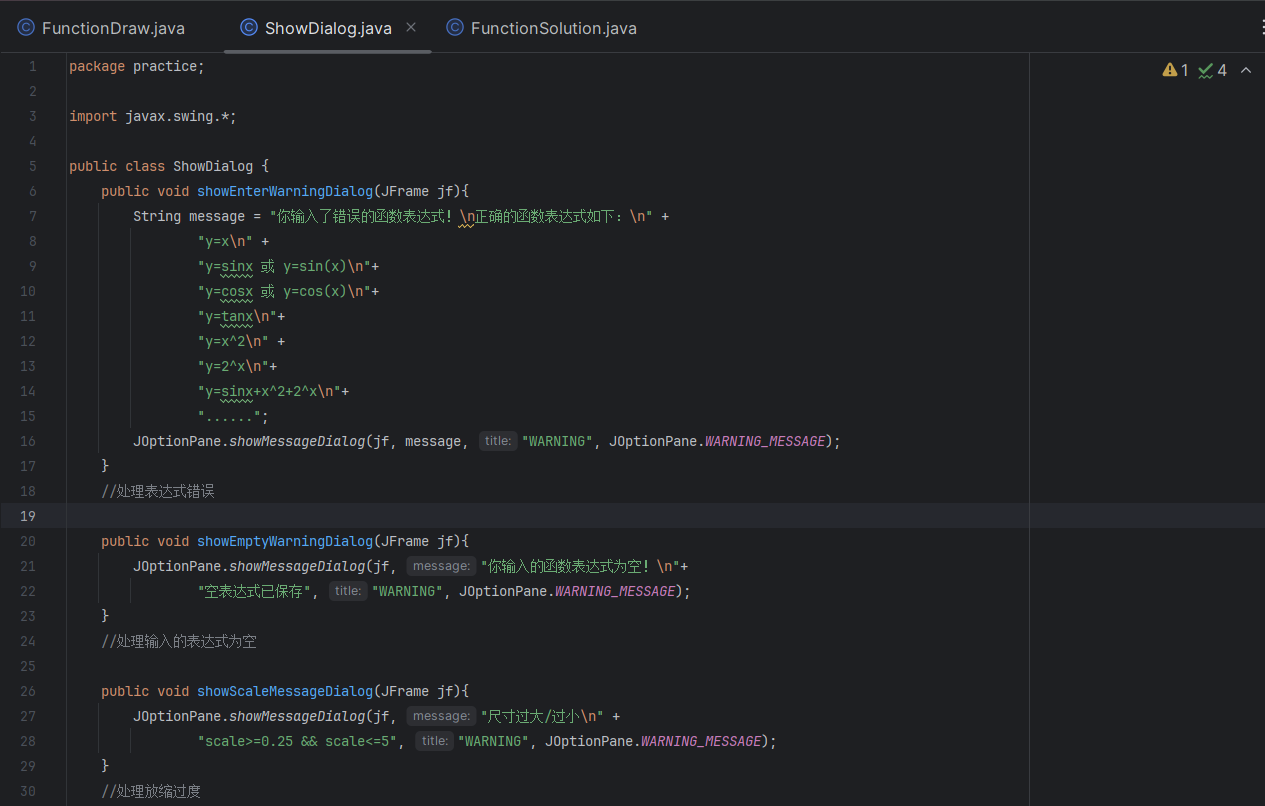
**保存的文件如下：（实现了日期和时间来划分文件夹）：**



**打开的显示内容如下：**

**部分代码截图：**

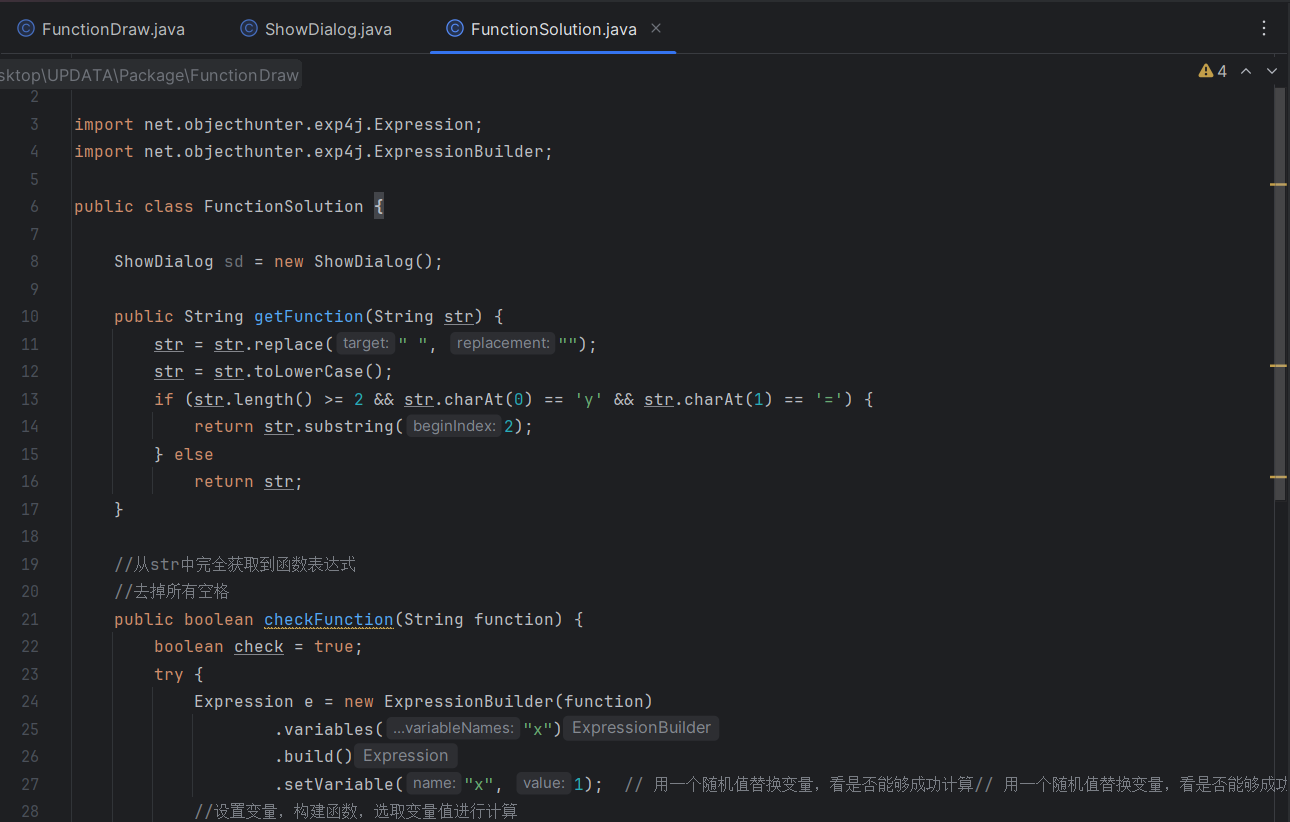
关于ShowDialog类：





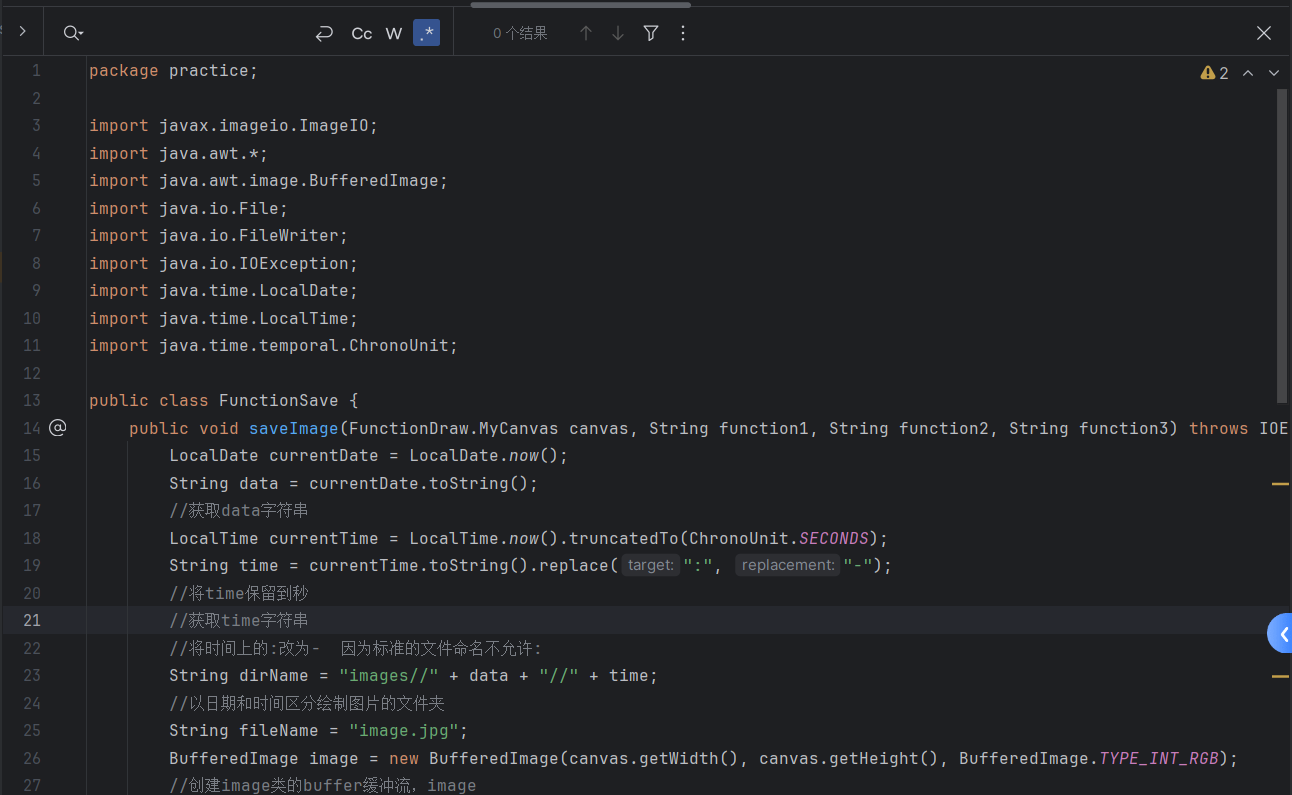


关于FunctionSolution类：

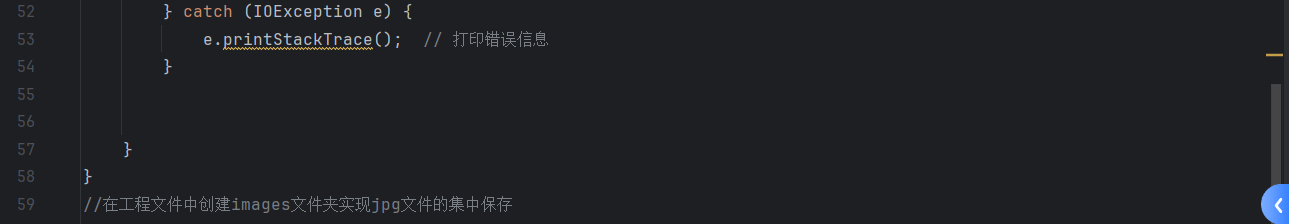






关于FunctionSave类：  








主类代码



