图形程序设计

Swing 用于表示“被绘制的”用户界面类，AWT 表示事件处理等窗口工具箱的底层机制。所有代码无条件加入

```
import java.awt.*;
import javax.swing.*;
```

顶层窗口（未被包含在其他窗口中的窗口）称为**框架（frame）**。在 Swing 中框架类为 JFrame，它不绘制在画布上，而是由窗口系统绘制。

类似下图的代码可创建窗口

```
public static void main(String[] args) {
  EventQueue.invokeLater(() -> {
    MyFrame frame = new MyFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  });
}
```

注意让框架显示需要调用 setVisible 方法。可以用 setLocation，setBound，setIconImage，setTitle，setResizable 方法设置其他属性。

Tookit 类包含许多于本地窗口系统打交道的方法，可以借助

```
Toolkit kit = Toolkit.getDefaultToolkit();
Dimension screenSize = kit.getScreenSize();
```

获得屏幕大小。

为了向 frame 中添加组件，需要向 frame 的**内容窗格（content pane）**中添加之，然而实际上可以直接添加为

```
frame.add(c);
```

为了定义 component 类，下列方法是必须的：

```
public void paintComponent(Graphics g) {
  g.drawString("Hello World!", x, y);
}
```

paintComponent 方法由系统自动调用，无需自行调用。如果需要刷新，应

调用 repaint 方法。Graphics 类型的对象 g 保留绘制图像和文本的设置，所有绘制都依赖于它。

同时还应当重写 getPreferredSize 方法，返回首选宽度和高度。

```
public Dimension getPreferredSize() {
  return new Dimension(w, h);
}
```

最后，不同于上述空框架的 setSIze，可以直接用 pack 方法将 frame 设置为满意的大小。

Graphics2D 提供了比 Graphics 更强大的绘图功能，可以通过下列方法调用

```
Graphics2D g2 = (Graphics2D) g;
Rectangle2D r = new Rectangle2D.Double(...);
g2.draw(r);
```

使用内部类 Double 是为了使用 Double 指定坐标。

Rectangle2D 和 Ellipse2D 在很大程度上类似，并且可以通过

```
Rectangle2D r = new Rectangle2D.Double();
r.setFrameFromDIagonal(px, py, qx, qy);
```

或者

```
Point2D p = ...; Point2D q = ...;
r.setFrameFromDiagonal(p, q);
```

从对角点设定大小。此外还有 Line2D 可供绘图使用。

```
g2.setPaint(Color.RED);
g2.draw(r);
g2.fill(r);
```

可以以给定的颜色绘图或填充。SystemColor.* 可获得预设颜色。

```
setBackground(color);
setOpaque(true);
```

可以设定背景。

通过

```
String[] fonts = GraphicsEnvironment
                .getLocalGraphicsEnvironment.
                .getAvailableFontFamilyNames();
```

可以获得字体集。也可以直接使用 SansSerif，Serif，Monospaced 等直接引用字体。通过

```
// 常规方法
Font f = new Font("SansSerif", Font.BOLD + Font.ITALIC, 14);
// 换字号
Font f2 = f.deriveFont(14.0F);
// 从文件获得
URL url = new URL("Arial.ttf");
InputStream in = url.openStream();
Font f3 = Font.createFont(Font.TRUETYPE_FONT, in);
// 设定字体
g2.setFont(f);
```

获得字体并且设定字体。

为了获得渲染字体的大小，需要字符串与绘制设备，执行

```
FontRenderContext context = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(message, context);
```

获得矩形。其坐标横轴为基线，纵轴为左边界，故

```
double width = bounds.getWidth(); // 宽
double height = bounds.getHeight(); // 高
double acsent = -bounds.getY(); // 上坡度

LineMetrics metrics = f.getLineMetrics(message, context);
float ascent = metrics.getAscent(); // 上坡度
float descent = metrics.getDescent(); // 下坡度
float leading = metrics.getLeading(); // 行距
```

为了绘制文本，应当获得基线左端点的坐标。

```
g2.drawString(message, x, upperLeftCornerY + ascent);
```

通过

```
Image image = new ImageIcon(filename).getImage();
g2.drawImage(image, x, y);
g2.copyArea(upperleftX, upperleftY, bottomrightX, bottomrightY,
    targetX, targetY);
```

获得并绘制、复制图像。

事件处理

AWT 处理事件的方式谓，从事件源到事件监听器的事件委托模型。事件源产生事件时，会向事件注册的所有事件监听器对象发送一个通告。

```
class MyListener implements ActionListener {
  public void actionPerformed(ActionEvent event) { ... }
}
ActionListener listener = ...;
button.addActionListener(listener);
// lambda表达式
button.addActionListener(event -> ...);
```

为了调整观感，可以借助

```
try {
  UIManager.setLookAndFeel(className);
  SwingUtilities.updateComponentTreeUI(frame);
  pack();
} catch (Exception e) { e.printStackTrace(); }
```

其中 className 的选择范围在

```
UIManager.LookAndFeelInfo[] infos =
    UIManager.getInstalledLookAndFeels();
```

对于有多个事件的 Listener 接口，可以通过继承其 Adapter 来获得同样效果，而无需将所有方法均重写。

```
class Terminator extends WindowAdapter {
  public void windowClosing(WindowsEvent e) { ... }
```

```
}
frame.addWindowListener(new Terminator());
```

可以通过继承 Action 或 AbstractAction 以实现将同一动作复用至多个事件源。

```
public class MyAction extends AbstractAction {
  public void ActionPerformed(ActionEvent event) { ... }
}
Action myAction = new MyAction(...);
JButton myButton = new JButton(myAction); // 添加给按钮
JMenuItem myItem = new JMenuItem(myAction); // 添加给菜单项
```

还可以通过设定

```
public myAction() {
  putValue(Action.NAME, name);
  putValue(Action.SMALL_ICON, icon);
}
```

改变按钮上的文字与图标。

为了获得键盘输入，先通过 InputMap 将子元素的键盘输入映射到动作键，再通过 ActionMap 将动作键映射到 Action。

```
InputMap imap =
    panel.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
imap.put(KeyStroke.getKeyStroke("control Y"), "panel.yellow");
ActionMap amap = panel.getActionMap();
amap.put("panel.yellow", yellowAction);
```

与鼠标点击有关的事件定义在 MouseAdapter 内，与鼠标移动、拖动有关的事件定义在 MouseMotionListener 内。

```
setCursor(Cursor.getDefaultCursor());
setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
```

可以修改光标样式。

除了上述各种 Listener，还有用于滚动条的 AdjustmentListener，用于复选框的 ItemListener 等。

各种控件

表格中的数据以二维数组的形式储存

```
Object[][] cells = { { "Alice", 92 }, { "Bob", 73 } };
```

标题以`String[]`的形式储存，后

```
JTable table = new JTable(cells, titles);
JScrollPane pane = new JScrollPane(table);
```

XML

为了读取 XML，通过

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
File f = ... ; // 文件方式
Document doc = builder.parse(f);
URL u = ... ; // 指定URL
Document doc = buidler.parse(u);
InputStream in = ... ; // 指定输入流
Document doc = builder.parse(in);
```

之后通过

```
Element root = doc.getDocumentElement();
String tagName = root.getTagname();
```

获得根节点与标签名。之后枚举

```
NodeList children = root.getChildNodes();
for ... {
  Node child = children.item(i);
  if (child instance of Element) Element childElement = (Element) child;
}
```

对于只包含文本的 Element，可以

```
Text textNode = (Text) childElement.getFirstChild();
```

```
String text = textNode.getData().trim();
```

为了获得属性，可以通过

```
NamedNodeMap attributes = element.getAttributes();
Node attribute = attributes.item(0);
String name = attribute.getNodeName();
String value = attribute.getNodeValue();
// 知道属性名可以直接获取
String unit = element.getAttribute("unit");
```