介绍

实际上我们学习的是JavaFX，但是Java的图形化编程的库很多，我们只是了解一些方法和类

下面的代码都是基于ipLee的一些项目

设计思想

这里介绍的都会是树形的布局方式，即元素位置都依托于其父结点

因为在初始化方面的考虑，我们使用下面的方式调用元素：

```
TextArea contentText ;
TextArea get_contentText(){
    if(contentText!=null){
        return contentText;
    }
    contentText = new TextArea();
    contentText.setPrefSize(Double.MAX_VALUE, Double.MAX_VALUE);
    contentText.setFont(new Font(20));
    contentText.setStyle("-fx-text-fill:black");
    contentText.setEditable(false);
    return contentText;
}
```

只用`get_contentText()`方法返回`contentText`实例，当`contentText`为`null`（也就是还没有初始化）时，顺便将其初始化

这种方式的好处在于：

1. 可以不关心元素的初始化顺序，因为需要在某一个元素的时候调用`get_XX`方法就行，如果正好这个元素没有被初始化，就可以顺便将其初始化
2. 可以将代码分成多个块，某一个内容的初始化就放在某一个方法内，可以方便写代码

JavaFX的整体写法

比如我们有一个JavaFX程序叫`Ball`，那么整体写法应该是这样的：

```
public class Ball extends Application {
    double width=1800;double height=1000;
    Circle ball;
    Circle get_ball(){
        if(ball!=null){
            return ball;
        }
```

```
        ball=new Circle();
        // ...
        return ball;
    }
    Group root;
    Group get_root(){
        if(root!=null){
            return root;
        }
        root=new Group();
        root.getChildren().add(get_ball());
        // ...
        return root;
    }
    @Override
    public void start(Stage window) throws Exception {
        window.setTitle("Ball");
        window.setScene(new Scene(get_root()));
        window.setWidth(width);window.setHeight(height);
        window.setResizable(false);
        window.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

重点：继承自`Application`、覆写了`start`方法，在`main`里调用`Application.launch(args);`

JavaFX的类和初始化示例

文本类

`TextArea`和`TextField`是两个文本类的类，都可以用来作为文本输入和展示。两者区别在于：`TextArea`允许多行（甚至会自动帮你生成滚动条），而`TextField`不允许（即使是`setText`设置了一个带`'\n'`的字符串也不行）

`TextArea`

```
TextArea contentText ;
TextArea get_contentText(){
    if(contentText!=null){
        return contentText;
    }
    contentText = new TextArea();
//  contentText.setPrefSize(Double.MAX_VALUE, Double.MAX_VALUE); // 可以用来设置初始大小
    contentText.setFont(new Font(20)); // 字体大小
    contentText.setStyle("-fx-text-fill:black"); // 颜色
    contentText.setEditable(false); // 可否编辑，默认为否
    return contentText;
}
```

`TextField`

```
TextField inputText ;
TextField get_inputText(){
   if(inputText!=null){
      return inputText;
   }
   inputText = new TextField();
   inputText.setFont(new Font(10)); // 字体大小
   inputText.setPrefSize(250, 20); // 初始大小
   inputText.setOnKeyPressed(e -> { // 按回车发送的动作
      if (e.getCode().equals(KeyCode.ENTER)) {
         get_contentText().setText(get_contentText().getText() +'\n'+
get_inputText().getText());
         get_inputText().setText(""); // 发送完之后清零
      }
   });
   return inputText;
}
```

按钮类

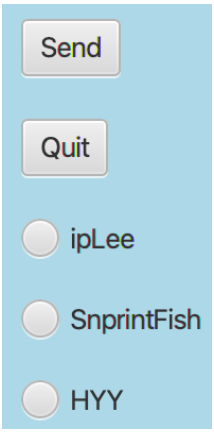
主要是看怎么设置按钮动作

```
Button buttonSend;
Button get_buttonSend(){
    if(buttonSend!=null){
        return buttonSend;
    }
    buttonSend = new Button("Send"); // 按钮的名称，按钮上会有'Send'的字符串
//  buttonSend.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
    buttonSend.setOnAction(evt -> { // 按下之后的动作
        contentText.setText(get_contentText().getText() + get_inputText().getText());
        get_inputText().setText(" ");
    });
    return buttonSend;
}
```

VBox和按钮组

这个是纵向布局工具，其子结点会自动被排列成一列

这里以我的聊天室控制栏为例：




```
VBox buttons;
String[] connectingMemberNames={"ipLee","Snprint_Fish","HYY"}; // 三个空间的名字
VBox get_buttons(){
    if(buttons!=null){
        return buttons;
    }
    buttons = new VBox(20);
    buttons.setStyle("-fx-background-color:lightblue"); // 设置背景颜色
//      buttons.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE); // 设置最大大小，虽然好像不设置也没问题
    buttons.setPadding(new Insets(10, 10, 30, 30)); // 设置和边界的距离

    // 将上面的两个按钮加入，注意VBox的Children加入顺序就是排列从上往下的顺序
    get_buttons().getChildren().add(get_buttonSend());
    get_buttons().getChildren().add(get_buttonQuit());
```

```
    RadioButton [] radioButtons=new RadioButton[connectingMemberNames.length]; // 下面三个按钮的数组
    for (int i=0;i<radioButtons.length;++i){
        radioButtons[i]=new RadioButton(connectingMemberNames[i]); // 使用三个名字作为按钮的标识
    }
    for (RadioButton radioButton : radioButtons) { // 实现“按了A，BC都消失”的效果
        radioButton.setOnAction(evt -> {
            if (radioButton.isSelected()) { // 如果该按钮已经被选中
                for (RadioButton button : radioButtons) { // 对于所有按钮
                    if (button != radioButton) { // 如果不是刚刚被选中的按钮
                        button.setSelected(false); // 就让它不被选中
                    }
                }
            }
        });
    }
    get_buttons().getChildren().addAll(radioButtons); // 将数组内容加入Children
    return buttons;
}
```

菜单栏和菜单及显示内容

`Menu`和`MenuBar`实现的

```
Menu menuHelp; // help的menu
Menu get_menuHelp(){
    if (menuHelp!=null){
        return menuHelp;
    }
    menuHelp = new Menu("Help"); // 用'Help'作为名字
    menuHelp.getItems().add(new MenuItem("About"){ // 添加名为'About'的菜单子项
        { // 设置这个菜单子项应该有怎么样的性质
            setOnAction( (e) -> { // 如果这个菜单子项被选择
                Stage stage = new Stage(); // 就新建一个窗口
                stage.setTitle("About us"); // 新窗口的标题是'About us'
                stage.setScene(new Scene( // 将新窗口的内容设为一个新的Scene
                                          new BorderPane(){ // 新的Scene只有一个BorderPane作为展示
                        {
                            setPadding(new Insets(2,2,2,2)); // 设置BorderPane和边框的距离
                            setCenter(new TextArea(){ // 设置BorderPane的中心为新的TextArea，其内容为新的TextArea
                                {
                                    setText("点击右侧单选按钮，可以选择不同的人\n" +
                                            "菜单栏与聊天框字的颜色是粉色的，字号为20\n" +
```

```
                                                "右侧两个按钮可以选择Quit和Send\n"); // 新的
TextArea内容为左
                                        setFont(new Font(20)); // 新的TextArea的字体大小为20
                                    }
                                });
                            }
                        }));
                stage.show(); // 显示窗口
            });
        }
    });
    return menuHelp;
}
Menu menuFile;
Menu get_menuFile(){
    ... // 类似实现，不展示了
}
MenuBar menuBar;
MenuBar get_menuBar(){
    if(menuBar!=null){
        return menuBar;
    }
    menuBar = new MenuBar();
    menuBar.setStyle("-fx-background-color:grey"); // 将menuBar设置成灰色
    menuBar.getMenus().addAll(get_menuFile(), get_menuHelp()); // 将两个menu加入menuBar
    return menuBar;
}
```

`BorderPane`作为根

对于需要自由调整大小的布局，可以使用`BorderPane`作为根

```
BorderPane root;
BorderPane get_root(){
    if(root!=null){
        return root;
    }
    root = new BorderPane();
    root.setPadding(new Insets(2, 2, 2, 2)); // 设置边框距离
    root.setCenter(get_contentText()); // 将中心设置成contentText
    root.setTop(get_menuBar()); // 将顶部设置成menuBar
    root.setRight(get_buttons()); // 将右段设置成buttons的VBOX
    root.setBottom(get_inputText()); // 将底部设置成inputText
    return root;
}
```

`GridPane`作为根

这里展示的是我计算器程序按钮部分的代码，像按钮这种布局还比较适合`GridPane`

```
GridPane buttonsPane;
GridPane get_buttonPane(){
    if(buttonsPane!=null){
        return buttonsPane;
    }
    buttonsPane=new GridPane();
    buttonsPane.setHgap(4);buttonsPane.setVgap(4); // 将格点设置成4*4的
//      buttonsPane.getChildren().addAll(get_buttons().values());
    Map<String,Pair<Integer,Integer>>position=new HashMap<String,Pair<Integer,Integer>>
(){
        {
            put("7",new Pair<Integer,Integer>(0,0));
            put("8",new Pair<Integer,Integer>(1,0));
            put("9",new Pair<Integer,Integer>(2,0));
            put("4",new Pair<Integer,Integer>(0,1));
            put("5",new Pair<Integer,Integer>(1,1));
            put("6",new Pair<Integer,Integer>(2,1));
           ... // 这个是记录某个按钮的text列和行对应的Pair的key、value的元素
    };
    for (Button button:get_buttons().values()){ // get_buttons是获得Map<String,Button>,
这里遍历所有的Button
        buttonsPane.add(button,
            position.get(button.getText()).getKey(),
            position.get(button.getText()).getValue()
        );
        // 相当于add(button,c,r)，其中button是要加入的元素，c,r是元素的列和行
    }
    buttonsPane.setMaxSize(Double.MAX_VALUE,Double.MAX_VALUE); // 设置最大大小（虽然好像没啥
用）
    buttonsPane.setPrefSize(width,height/5.2*4); // 设置宽度和高度
    return buttonsPane;
}
```

`Group`作为根

在需要自行排版的时候可以使用`Group`作为根，比如：

```
Button get_nextButton() {
    if(nextButton!=null){
        return nextButton;
    }
    nextButton = new Button();
```

```
    ...
    nextButton.setTranslateX(width / 2);
    nextButton.setTranslateY(height - 50); // 在使用Group作为根的时候需要使用setTranslate设置位置
    return nextButton;
}
// in start(Stage window)
root = new Group();

root.getChildren().add(get_nextButton());
init_bigCircle();
root.getChildren().add(bigCircle); // 添加子结点的方式
...

Scene scene = new Scene(root, width, height); // root作为显示内容，宽width，高height
window.setScene(scene);
window.setTitle("AVLTree-Demo"); // 名字
// window.setResizable(false); // 不可变化大小
window.show();
```

最终显示

```
// window是start传入的Stage window
window.setTitle("Calculator"); // window的title
window.setScene(new Scene(get_root())); // 设置Scene，其只有root一个儿子结点，这样方便控制
window.setWidth(width);window.setHeight(height); // 设置大小
window.setResizable(false); // 调整是否可改变大小
window.show(); // 显示
```

JavaFX的图像和动画

以我写的AVLTree的为基础，进行删减

直线动画

方法目的：将`bigCircle`直线移动

注意，这是单步动画，需求是“点一下进行一段动画”时这么写

```
static void moveLightCircle(int p) {
    Line line = new Line(bigCircle.getCenterX(), bigCircle.getCenterY(), // line的起点:
bigCircle当前位置
            nodeBase[p].getCenterX(), nodeBase[p].getCenterY()); // line的终点，这里不用关
心nodeBase[p]的意义
    PathTransition path = new PathTransition(new Duration(duration), line, bigCircle);
// 动画PathTransition
    // new Duration(duration)为新的Duration作为动画持续时间的设置，line为动画轨迹，bigCircle为
动画对象
    // 注意PathTransition只是进行了一次移动动画，并没有改变bigCircle的位置，所以需要重新调整
bigCircle的位置
    bigCircle.setCenterX(line.getEndX()); // 将bigCircle调整到line的终点X,Y坐标
    bigCircle.setCenterY(line.getEndY());
    path.setCycleCount(1); // 只进行一次动画循环
    path.play(); // 播放动画
}
```

将`Line`改成其他的比如`Circle`就可以获得圆形轨迹

`Circle`图形

```
Circle bigCircle ;
void init_bigCircle(){
   bigCircle= new Circle(width / 2, height / 2, 30); // 位置，半径
   bigCircle.setFill(Color.TRANSPARENT); // 设置球的中心什么颜色，这里设置成了透明
//   ball.setFill(Color.valueOf("grey")); // 这是一种方式
   bigCircle.setRadius(radius * 1.5); // 设置半径
   bigCircle.setStroke(Color.RED); // 设置bian ku a
   bigCircle.setStrokeLineJoin(StrokeLineJoin.ROUND); // 设置描边类型
   bigCircle.setVisible(true); // 设置可见性，默认可见
   bigCircle.setCenterX(...);
   bigCircle.setCenterY(...); // 设置位置
}
// 在后面将bigCircle装入root然后window.show()就能显示了
```

`Rectangle`图形

```
Rectangle rectangle;
Rectangle get_rectangle() {
    if (rectangle != null) {
        return rectangle;
    }
    rectangle = new Rectangle(width / 1000, height / 1000, Color.valueOf(color)); //
width,height,Fill
//    rectangle.setX();
//    rectangle.setY();
    return rectangle;
}
```

异步单步动画

在某时某刻可能需要有多个动画同时进行，比如多个球同时运动，可以使用`Queue<Runnable>`实现需求。实际上就是将多个球的运动操作同时`add`到一个队列里，然后在某一时刻将所有操作从队列中取出，然后`run()`。由于从队列取出几乎不用耗时间，所以表现上就像所有球一起`play()`一样

比如（以AVLTree可视化为基础）

```
// 这里有一个static的问题，不过我们就不管了。。。
// in DemoTree
static Queue<Runnable>actions=...;
static void moveNode(int ps, int pt) { // 一个moveNode操作动画方法
    Circle baseCps = ...;
    Circle baseCpt = ...;
    Line line = new Line(baseCps.getCenterX(), baseCps.getCenterY(), // 起点的位置
            baseCpt.getCenterX(), baseCpt.getCenterY()); // 终点的位置

    PathTransition path = new PathTransition(new Duration(duration*3), line,
stackPane); // 动画时间，轨迹，移动操作对象
    stackPane.setTranslateX(baseCpt.getCenterX()); // 改变移动对象最后位置
    stackPane.setTranslateY(baseCpt.getCenterY());
    path.setCycleCount(1); // 只进行一次展示
    path.play(); // 异步方法
}
static Button get_nextButton() { //
    nextButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            if (!actions.isEmpty() ) {
                Objects.requireNonNull(actions.poll()).run(); // run()实际上就会调用
Runnable存储的moveNode方法
                // Queue的poll是返回队首元素并且推出队首元素；peek是只返回队首元素但是不推出
            }
        }
    });
```

```
    ...;
    return nextButton;
}
// in AVLTree
int _moveNode(int ps, int pt) { // 将ps的val移动到pt
    ...;
    DemoTree.actions.add(new Runnable() { // 这里在队列里添加了一个调用.run()就会产生移动某一个
结点动画的Runnable对象
        @Override
        public void run() {
            DemoTree.moveNode(ps, pt);
        }
    });
    return cnt;
}
void _moveTree(int ps, int pt) { // 将ps的子树交换到pt，也就是多个结点移动
    for (int l = 1; l + lp <= 8; ++l) {
        int _2l = 1 << (l - 1);
        for (int i = 0; i < _2l; ++i) {
            _moveNode(_2l * ps + i, _2l * pt + i);
        }
    }
}
```

连续动画设计`AnimationTimer`

这里指去年的期末的题：

五、(15 分) 当前目录存在一个文件“ball.dat”，文件存储了弹跳球信息，存储了弹跳的参数（double x, double y, double speedX, double speedY, String color）。
编写程序，满足以下要求：
（1） 读取文件“ball.dat”
（2） 无需考虑碰撞
（3） 生成 1 个弹跳球 JAVAFX 可视化展示弹跳球运动

这个~~很明显~~有一种设计思路：

1. 让球沿着当前位置移动
2. 检测球有没有撞壁
    1. 没撞：跳过
    2. 撞了：改变`speedX`和`speedY`

这里介绍`AnimationTimer`类：这是一个抽象类，它有一个`handle`方法需要被实现。它的作用是：在调用`start()`后，播放每一帧都会调用一次`handle()`，见下面的我写的答案

```
package Ball;

import javafx.animation.AnimationTimer;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;

import java.util.LinkedList;
import java.util.Queue;

public class Ball extends Application {
    double width = 900;
    double height = 500;
    double radius = 30;
    String color;

    void init_data() {
        dX = 2;
        dY = 1;
        color = "grey";
    }


    Circle ball;
    Circle get_ball() {
        if (ball != null) {
            return ball;
        }
        ball = new Circle(width / 2, height / 2, radius, Color.valueOf(color));
        ball.setVisible(true);
        // ...
        return ball;
    }

    Rectangle rectangle;
    Rectangle get_rectangle() {
        if (rectangle != null) {
            return rectangle;
        }
```

```
        rectangle = new Rectangle(width / 1000, height / 1000, Color.valueOf(color));
//      rectangle.setX();
        return rectangle;
    }

    Group root;
    Group get_root() {
        if (root != null) {
            return root;
        }
        root = new Group();
        root.getChildren().add(get_ball());
        System.out.println(root.getChildren());
        // ...
        return root;
    }

//  AnimationTimer anime;
//  AnimationTimer get_anime() {
//      if (anime != null) {
//          return anime;
//      }
//      anime = new AnimationTimer() {
//          @Override
//          public void handle(long now) {
//              double X = get_ball().getCenterX(), Y = get_ball().getCenterY(), R =
get_ball().getRadius();
//              if (X + R >= get_scene().getWidth()
//                      || X - R <= 0) {
//                  dX = -dX;
//              }
//              if (Y + R >= get_scene().getHeight()
//                      || Y - R <= 0) {
//                  dY = -dY;
//              }
//              get_ball().setCenterX(X + dX);
//              get_ball().setCenterY(Y + dY);
////                System.out.println(get_ball());
////                System.out.println(get_scene().getWidth() + " , " +
get_scene().getHeight());
//          }
//      };
//      return anime;
//  }

    double duration = 5;
    double dX, dY;
    Timeline anime;
    Timeline get_anime() {
```

```
        if (anime != null) {
            return anime;
        }
        anime = new Timeline(new KeyFrame(new Duration(duration), new
EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                double X = get_ball().getCenterX(), Y = get_ball().getCenterY(), R =
get_ball().getRadius();
                if (X + R >= get_scene().getWidth()
                        || X - R <= 0) {
                    dX = -dX;
                }
                if (Y + R >= get_scene().getHeight()
                        || Y - R <= 0) {
                    dY = -dY;
                }
                get_ball().setCenterX(X + dX);
                get_ball().setCenterY(Y + dY);
//              System.out.println(get_ball());
//              System.out.println(get_scene().getWidth() + " , " +
get_scene().getHeight());
            }
        }));
        anime.setCycleCount(Timeline.INDEFINITE);
        return anime;
    }

    Scene scene;
    Scene get_scene() {
        if (scene != null) {
            return scene;
        }
        scene = new Scene(get_root(), width, height);
        return scene;
    }


    @Override
    public void start(Stage window) throws Exception {
        init_data();
        window.setTitle("Ball");

        window.setScene(get_scene());
//      window.setWidth(width);window.setHeight(height);
//      window.setResizable(false);
        window.show();
//      get_anime().start();
        get_anime().play();
```

```
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

Timeline

在老师课件里找到的，使用方法如下：

```
double duration = 5;
Timeline anime;

Timeline get_anime() {
    if (anime != null) {
        return anime;
    }
    anime = new Timeline(new KeyFrame(new Duration(duration), new
EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) { // #
            if (X + radius >= width || X - radius <= 0) {
                dX = -dX;
            }
            if (Y + radius >= height || Y - radius <= 0) {
                dY = -dY;
            }
            X += dX;
            Y += dY;
            ball.setCenterX(X);
            ball.setCenterY(Y);
        }
    }));
    anime.setCycleCount(Timeline.INDEFINITE);
    return anime;
}
```

在使用`get_anime().play()`之后，每`duration`毫秒，就会调用一次`#`的`handle()`方法（这里也可以传入`lambda`函数）

另外也可以`setCycleCount`表示进行多少次调用，可以传入`Timeline.INDEFINITE`表示调用无数次