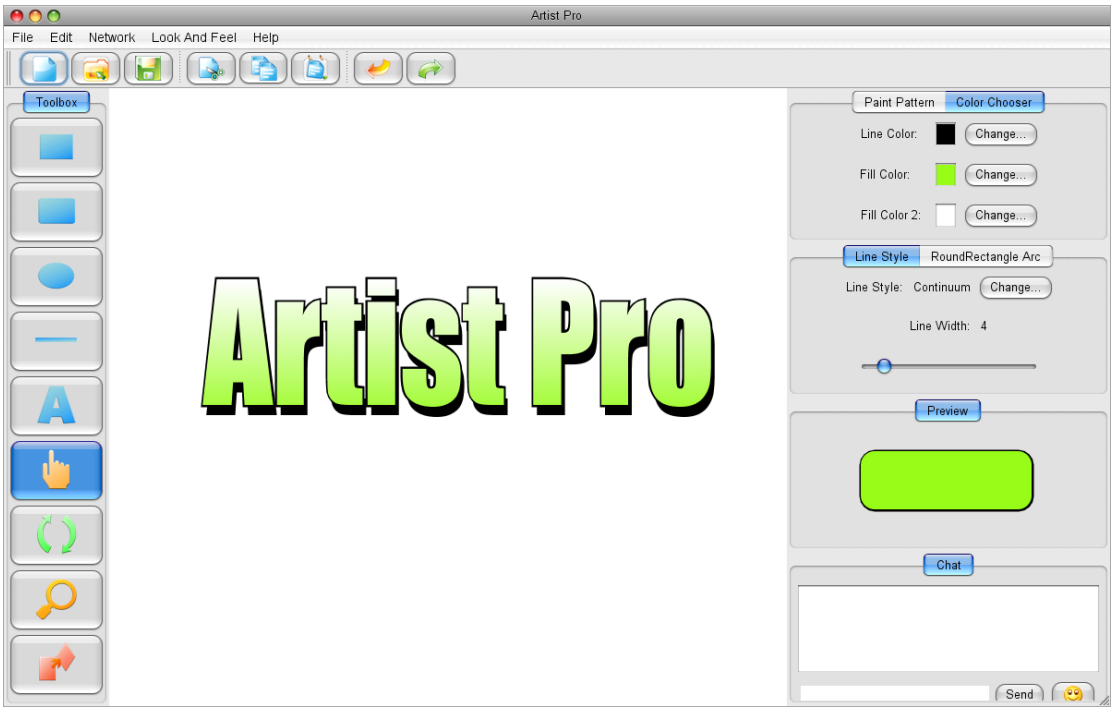
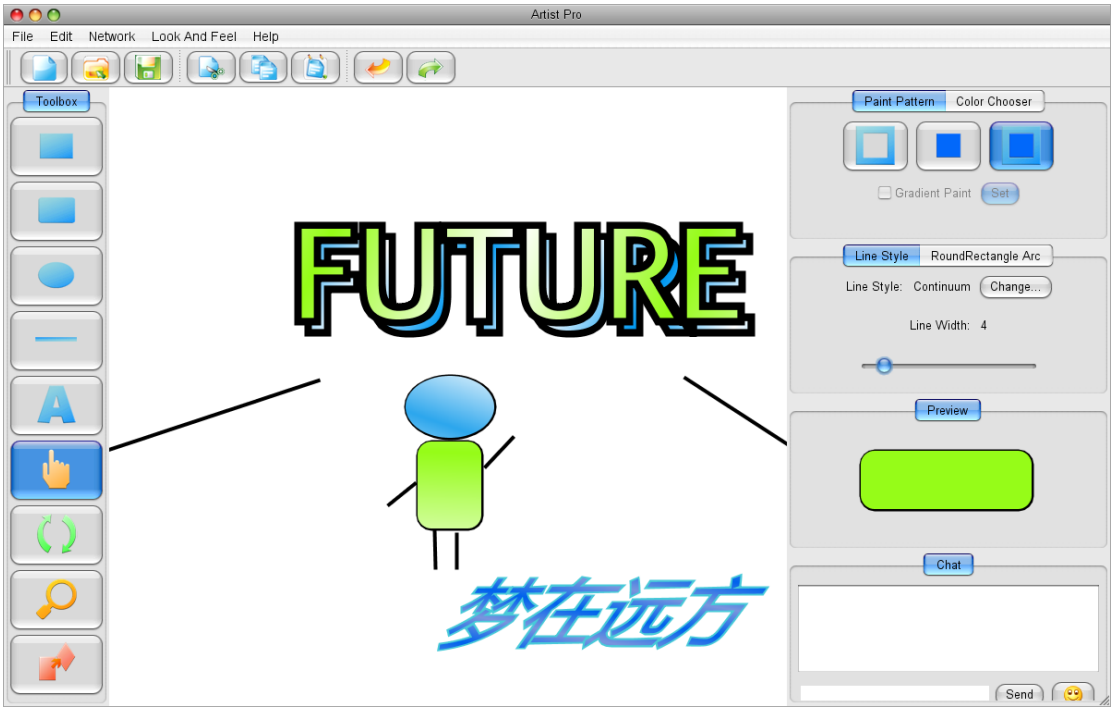
JAVA 程序设计与训练

实验报告

2005010130 计 64 侯杰



Artist Pro 程序运行界面



一、实验要求

1. 需要提供一个菜单或者工具栏，来体现程序的功能。

2. 能提供基本矢量图形和文字的绘制。只要求绘制直线、矩形、圆角矩形、椭圆以及文本。

3. 需要能对绘制完毕的图形再进行修改。需要支持一些几何变形：缩放、旋转和扭曲。

4. 需要提供图形的选择功能

5. 需要支持图形的填充色以及线条颜色的修改（直线仅有线条颜色），以及线条粗细调整。

6. 对于图片中的文字，需要提供修改字体的功能。

7. 提供图像的保存、读取功能。必须保证图片保存后再读取时结果一致。

8. 图形的绘制需要进行网络同步

9. 当两个人同步之后，需要提供聊天的功能。

在此基本要求之外，还可以自己增加扩充功能。

二、分析思路

总体来说，本次实验可以分为以下几个部分：绘图部分，编辑部分，文件处理部分和网络传输部分。下面分别针对每个部分进行分析。

（1）绘图部分

此部分要求几种简单几何体的绘制，通过查阅 javadoc 中的 API 可知，这些几何体的绘制都可以由 Graphics2D 类里面相应的函数来绘制。而如何给这些

函数设置需要的参数则需要从鼠标操作中来得到。基本过程为:

画布监听鼠标操作——>产生鼠标操作坐标——>计算出图形参数——>传递给 Graphics2D 的相应函数——>在画布上绘制出图形。

（2）编辑部分

编辑需要将画好的图形的某些属性重新赋值。刚画出的图形是具有默认属性的图形 ,若要编辑某个图形需要先提供图形的选取功能 ,获得此图形对象的引用，然后修改其属性。基本过程为:

画布监听鼠标操作——>选取图形——>画布监听鼠标操作——>产生鼠标操作坐标——>根据当前功能模式采取相应算法算出新的图形属性参数——>更新图形对象属性值。

（3）文件处理部分

编辑的好的图形需要写成文件存入外存中。由于当前内存中存的是图形对象，可以采用序列化的方式加上文件输出流存成文件。打开过程则相反。基本过程为：

内存中图形对象——>序列化——>文件输出流——>文件

内存中图形对象<——序列化<——文件输入流<——文件

（4）网络传输部分

网络部分为实现多人同步数据传输 ,采用一 server ,多 client 的连接形式。当 server 有信息需要发送时，就发送给所有连接到它的 client。当 client 有信息需要发送时，先将信息发送至 server，再由 server 转发至所有 client。基本过程为：

server 主程序信息发送请求——>server 信息发送线程——>所有 client 信息接收线程——>所有 client 主程序

client 主程序信息发送请求——>client 信息发送线程——>server 信息接收线程——> server 主程序信息发送请求——>server 信息发送线程——>所有 client 信息接收线程——>所有 client 主程序

三、体系架构

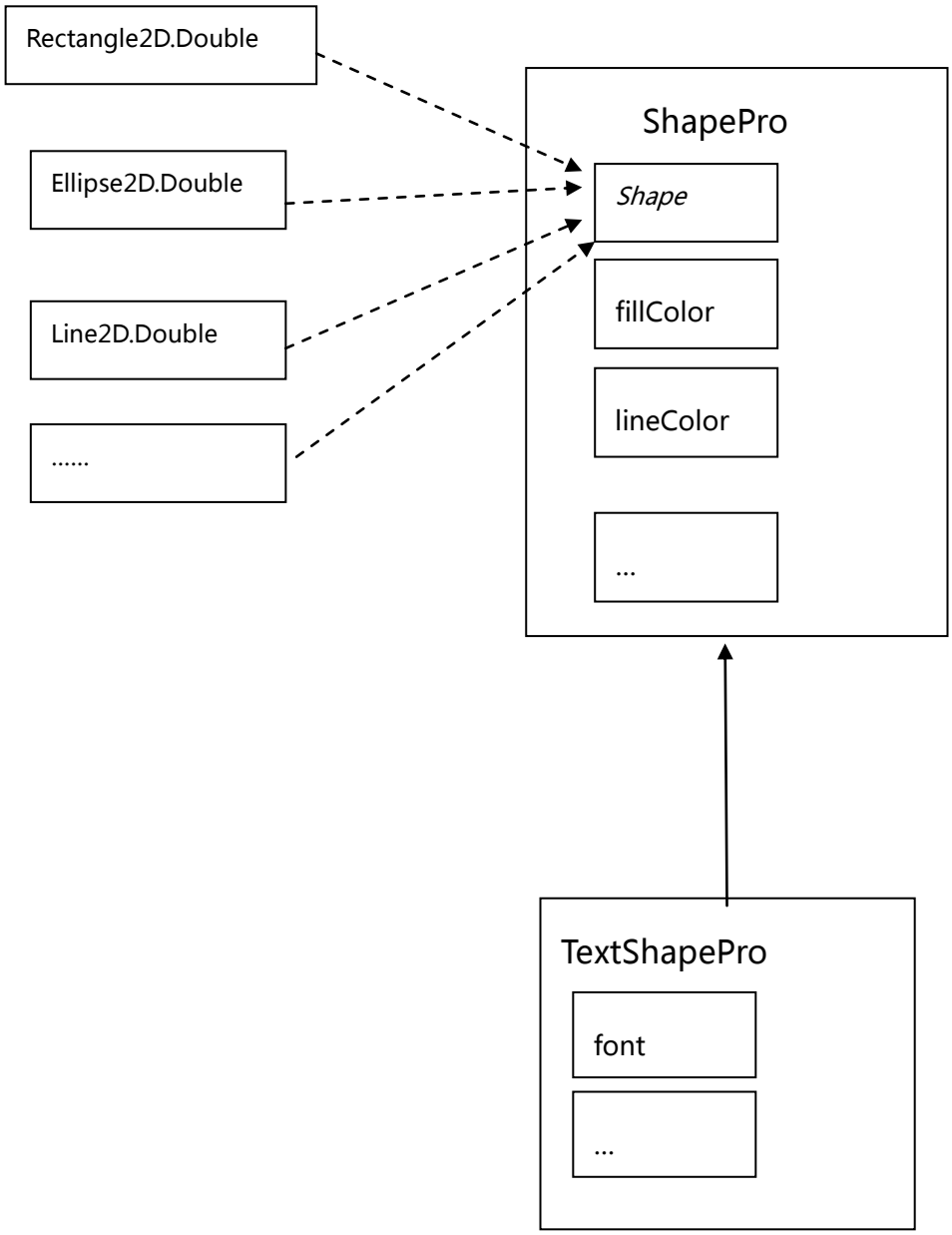
在这里我只说一下我的特色部分，其余部分的架构基本上都用的是 java 提供好的现成的架构。

（1）图形对象架构

查阅 javadoc 可知，所有基本几何图形都实现了 Shape 接口，对于文字，也可以通过库函数根据其边界信息产生实现 Shape 接口的对象实例。但是 Shape 接口只定义了图形的形状位置信息，缺少颜色，笔划，填充样式，变形参数等其他重要信息。于是我决定采用封装的形式，自己写一个 ShapePro 类。里面包含了 Shape 类型的引用，这样等于说 ShapePro 类可以包含任意简单几何图形的引用，同时定义了边线，填充，变形等多种属性(详见本程序 javadoc)。绘图时候，根据 ShapePro 里的各项参数可以自定义绘制 Shape 时的样式。

对于文字对象，也可以产生它的 Shape。不过文字对象有一些额外的属性，如文字内容，字体等属性。所以写了一个 TextShapePro 继承自 ShapePro 以支持文字更多属性的保存。

图形各类之间的关系图如下：



（2）网络模型架构

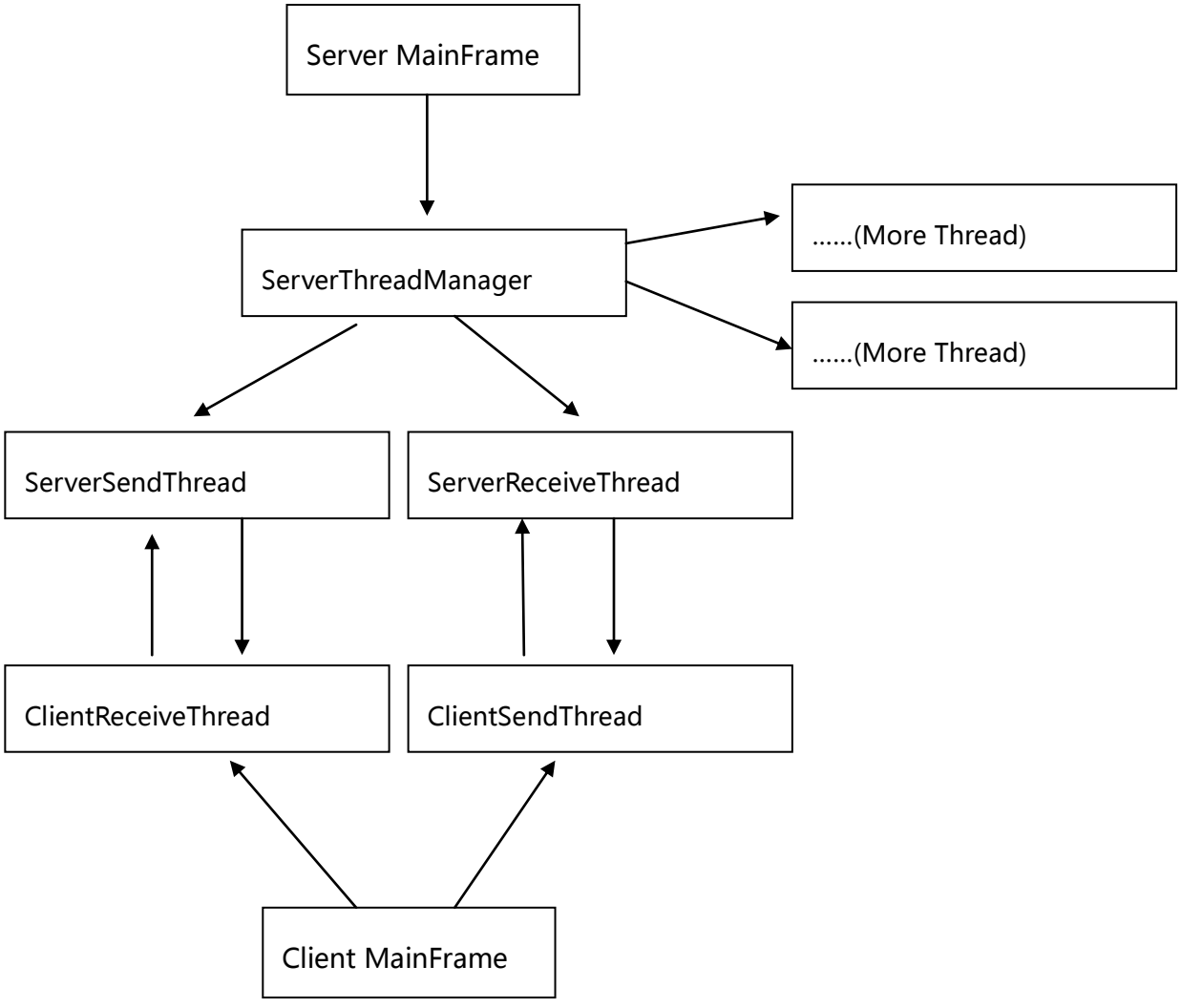
由于采用一 server，多 client 的形式，必定涉及到多线程的调度。网络部分由独立的现成完成。本次实验中，主程序和网络部分之间基本上是分离的，只

有图形对象信息和聊天文本信息在之间传递。

server 主程序进行建立服务器操作时，先开一个 ServerThreadManager 的线程，架构好 ServerSocket 并开始监听是否有客户端连接。若有一个新的客户来连接，则新开启两个线程 ServerSendThread 和 ServerReceiveThread 进行数据的发送和接收。若有 n 个用户连接 server，server 就要开总共 2n 个收发线程。

客户端进行连接服务器操作时，开启两个线程 ClientSendThread 和 ClientReceiveThread 进行数据的发送和接收。

模型如下：



四、功能实现

（1）新绘图形

所有的图形都是 ShapePro 类的对象，所有的图形都存储在 ArrayList<ShapePro> 的对象 ShapeProArray 中。

对于属性有两种，系统默认属性与图形自身属性。图形自身属性存储在 ShapePro 对象中，决定自身绘制的样式。系统默认属性存为 PaintProperty 的对象，主程序中留有名为 currentPaintProperty 的对其的引用，用来决定新绘图形的初始属性值。

软件有多种绘图模式，包括矩形模式，圆角矩形模式，椭圆模式，直线模式，文字模式。通过左侧工具箱摁钮选择不同的工作模式。主程序中有变量 currentFunction 来记录当前的工作模式。

鼠标在画布上摁下时，DrawHandler 中的 mousePressed 方法就根据当前的绘图模式立即生成一个相应类型的 ShapePro 对象，坐标为当前坐标，大小为 0，属性为系统默认值，并加入到 ShapeProArray 中。当继续拖动鼠标绘制时，mouseDragged 方法根据鼠标的操作，同时根据当前绘图模式不同，采用不同的算法，赋给刚新建的图形对象新的属性值。比如画矩形时，鼠标 press 一瞬间的坐标赋给矩形对象的 x,y 变量，鼠标在 x,y 方向移动的距离赋给矩形对象的 width,height 变量。当然，这仅是鼠标向右下方拖动时的算法。当鼠标向其他方向拖动绘制时，赋值操作会有相应的改动。

当鼠标 release 的一刻，绘图操作完成，对象赋值操作结束。

与绘制图形不同，绘制文字的过程是在画布上点鼠标后，在当前鼠标附近弹出一个对话框进行文字输入以及字体，字号，样式的选择。当输入完成点击对话框 ok 摁钮时，读入对话框的数据，生成新的 TextShapePro 对象，并把其 position 设为最开始鼠标点击的位置。

（2）显示图形

画布（canvas）继承自 JPanel，在其上进行所有的绘制操作。通过以上操作后，ShapeProArray 中已经加入了新的 ShapePro 对象，然后调用 canvas.repaint 方法，此方法再调用其 paint 方法绘制图形。所以重点在于重写 paint 方法。

paint 方法执行时，先进行一下一般的初始化操作，如设置渲染质量等。然后开始扫描 ShapeProArray，对其中的每一个 ShapePro 对象进行分析，将对象中的 stroke,fillColor,lineColor 等信息读出并通过 setColor，setPaint 等方法设置好绘图属性。然后读入 ShapePro 里面的关于变换的参数值。建立新的 AffineTransform 类对象并赋值。再读入 ShapePro 内的 Shape 对象，利用 AffineTransform 对象的方法将原 Shape 进行仿射变换成新的 transformedShape，再 draw 轮廓，最后 fill 填充。在 ShapePro 中有参数设定该图形是否有边线，是否有填充，或者填充的类型是 Solid 还是 Gradient。paint 方法执行时会根据这些数据决定以上绘制操作的取舍选择。

在一个图形绘图完成后，还会根据这个图形是否被选中来决定是否在图形边框上画上表示选中的虚线框，以及进行矢量控制的控制点。若有的话在图形区域的四角绘制。

对于文字对象，有特殊的处理方法。因为 TextShapePro 新建时里面的

Shape 引用并未指向任何 Shape 对象，所以要先得到 Graphics 对象的 FontRenderContext ,利用它新建一个 TextLayout 对象 ,最后利用 TextLayout 对象的 getOutline 方法返回一个实现 Shape 接口的对象实例，然后让 TextShapePro 中的 shape 引用指向它。这样文字也有了 Shape，就可以与其他图形采取同样的绘制方法。

当扫描完 ShapeProArray 后 ,本次绘图结束 ,等待下一次的 repaint 操作。

（3）编辑图形

编辑图形有两类。一种仿射变化编辑，另外一种是普通的属性修改编辑。对于后者，修改的基本上都是新绘图形时已经被赋值的量，比如图形位置，边线颜色，填充颜色，填充样式等。先介绍这种类型编辑。

选中所需要编辑的图形（选中过程在后面说），系统中一个引用 currentSelectedShapePro 就指向了所选中的对象，然后同时修改右侧属性面板的值，就对 currentSelectedShapePro 引用的对象的相应值进行更改，这样就可以同步修改当前选中的图形。如果未选中任何图形而更改属性面板，则修改的是系统默认绘图属性，即修改 currentPaintProperty。除了属性面板的编辑操作，拖拽操作也属于这类编辑操作。只不过这次是根据鼠标移动的量来决定如何修改原 Shape 对象的成员变量值。

另一种编辑操作比较复杂，即仿射变换。这些操作包括缩放，旋转和拉扯，需要在左侧工具栏选择相应的功能来进行操作。进行此类操作时，并不修改 Shape 对象的属性，而是修改 ShapePro 里面一些关于各类变换有无及变换的参数。这些参数在 canvas 的 paint 方法中读入并对 Shape 进行规定的仿射变换后输出。对于如何根据鼠标移动修改仿射变换的参数，不同变换，不同图形有着

不同的算法，在这里不一一赘述。但值得一提的是如果编辑的是一个已经具有仿射变换属性的图形，读取鼠标坐标后应先进行逆变换，变换回原来坐标系内的坐标值（因为人们总习惯是在新的变换后坐标系内操作，但我们要比如一个矩形旋转 45 度后，人们认为这回沿 45 度方向操作，应该和在矩形旋转前，沿水平方向操作具有相同的变形效果，而机器则认为这和原来是完全不同的操作，所以需要逆变换回原来的坐标系）。这样会有更好的操作手感，也更符合人的习惯。

（4）OFFICE 式矢量选取

要想选中图形，必须在图形内部点击鼠标才可以。在图形四角画上控制点，以后用户操作就必须把鼠标放在控制点才能操作（拖拽操作除外）。实现这个功能最重要的是对鼠标是否处于某个 Shape 内部的判断。这个操作可以使用 Shape 对象的 contains 方法进行判断，判断鼠标点击的位置是否处于区域中。当然这之前也需要将鼠标位置进行逆变换（用户会点击变换后的新区域，但是图形的仿射变换只是在输出时加的一层“滤镜”，实际图形的坐标位置并没有变，这样用户受变换结果干扰，就点不到本来的图形区域内了，所以要将鼠标位置逆变换）

（5）梯度填充

梯度填充是我增加的一个额外的功能。Graphich2D 中的 setPaint 方法可以设置填充为一个 GradientFill 的对象。一个 GradientFill 对象需要两个颜色及两个控制点信息。两个颜色在程序中分别叫做 fillColor1 和 fillColor2，其中 fillColor1 作为主填充颜色，而 fillColor2 则在梯度填充中作为第二填充颜色。颜色选择好后，由鼠标拖拽控制梯度填充。两个控制点分别由鼠标拖拽的 press 点和 release 点决定。最后产生一个新的 GradientFill 对象，赋给当前操作的

ShapePro 对象的相应值中。

（6）删除，剪切，复制，粘贴

删除最简单，只需在 ShapeProArray 中删除选中的 ShapePro 对象即可。复制操作将当前选中的图形复制一个副本给 clipboardShapePro，粘贴操作则是由 clipboardShapePro 再复制一个副本，并添加进 ShapeProArray 中。剪切可以看做是复制与删除的结合。

（7）撤销，重做

由于我并不知道每次操作的具体类型是什么（操作没有保存，只是保存了操作结果），所以进行撤销会比较麻烦。我采用了一种比较平凡的方法，即在每种会改变图形外观的操作执行前，都复制一个 ShapeProArray 的副本，并加入到 backShapeProArrayArray 中。这样 backShapeProArrayArray 随着操作一步步进行就装入了旧的图形信息。需要撤销时只需从它的最后取出一个 ShapeProArray，覆盖掉当前的 ShapeProArray 即可。

但注意为了实现重做功能，在撤销时我们不能直接覆盖掉当前的 ShapeProArray，而应该同样将它备份起来，我将它备份到 frontShapeProArrayArray 中。当执行重做功能时把 ShapeProArray 备份到 backShapeProArrayArray 中，然后再从 frontShapeProArrayArray 中取到刚才 Undo 时所放弃的数据，重新赋值给 ShapeProArray，这样就实现了重做的功能。

（8）网络多人同步

进行网络多人同步使用的是我在上文体系结构中所介绍的一 server 多 client 的方式。在网络上传输两种内容，一种是图形信息即 ShapeProArray，

它是一个 ArrayList<ShapePro>的对象。还有一个是聊天记录 chatRecord，它是 StringBuffer 的一个对象。两种对象都实现了 Serializable 接口，可以实现序列化。传输时只需使用 ObjectInputStream 和 ObjectOutputStream，一端 read，另一端 write 即可。

现在有两个问题需要解决。一是本次传来的到底是图形还是聊天表情。这个可以使用 instanceof 关键字来判断对象所属的类到底是那个。二是究竟什么时候要进行通信传数据。一般说来，只有一方图形发生更改才会传数据。所以我在主程序影响外观的图形更改操作后都进行了 setFlip 操作，其功能是将一个 int 变量的值由 1 到-1 或是由-1 到 1 进行跳变。发送线程也保存一个 Flip，一旦发现主程序的 Flip 与线程内部的 Flip 不等后，就开始传数据，然后将自己的 Flip 修改成与主程序的 Flip 一样的值。这样通过监测 Flip 就可以检测到主程序是否有发送数据的请求。

另外，server 接到 client 的数据后，要进行 setFlip 操作，这样服务器会再次将数据发向所有 client，实现一个 client 的数据向所有用户同步。而当 client 接收到 server 的数据后，不必进行 setFlip 操作，否则就会产生死循环，导致 server 与 client 之间传输永远不能停止。

（9）加入聊天表情

显示聊天内容的是一个 JTextPane，它有 insertIcon 方法可以插入图标。所以现在要解决的是如何让它知道应该在什么地方，插什么样的图标。我采用转义符输入的方法，定义以下转义符:

/smile　/angry　/haha　/bs

/cry　/faint　/bye　/omg

每个转义符代表一种聊天表情。用户只需输入转义符，系统调用 ChatTextAnalyzer 来进行转义符的翻译，并指导 JTextPane 在适当位置插入表情图片（gif 文件）。为了方便用户使用，又单独做了一个对话框，可以直接选择聊天表情，由系统自动输入转义符。

（10）文件处理

保存文件同样使用序列化，通过对象流，文件流输出至文件。打开文件的过程相反，不再赘述。

（11）导出成位图格式

为方便其他程序使用编辑的成果，增加了保存为位图的功能。可以保存为 JPG，BMP 和 PNG 三种格式。实现原理很简单，新建了一个 BufferedImage，在其上将画布上的内容重画一遍，然后用 ImageIO 类的 write 方法写入文件流即可。

五、实验后感想

此次 Java 大实验是我第一次亲手制作如此大型的程序。代码量大概近 6000 行，花了一个多星期的时间才完成。其间遇到无数次大大小小的障碍，更遇到过各种各样的 bug。不过功夫不负有心人，终于在规定时间内完成了，感到很有成就感。

通过这次大实验，我很快地熟悉了利用 Java 开发程序的一般方法，掌握了常见库函数的使用。同时对一般软件的设计模式也有了初步思考，进行了简单的试验式的架构工作。收获是相当大的。最重要的是，我锻炼了一个人不怕困难，独立思考的能力，培养了认真踏实的编程作风。这个项目时间实在是很宝贵的经

验。

最后感谢许斌老师，感谢给我们判作业、查实验的助教，谢谢你们的辛勤工作！

中秋快乐！

侯杰 2008.9.14