

Image Caption 协同标注平台

Scarlet Pan

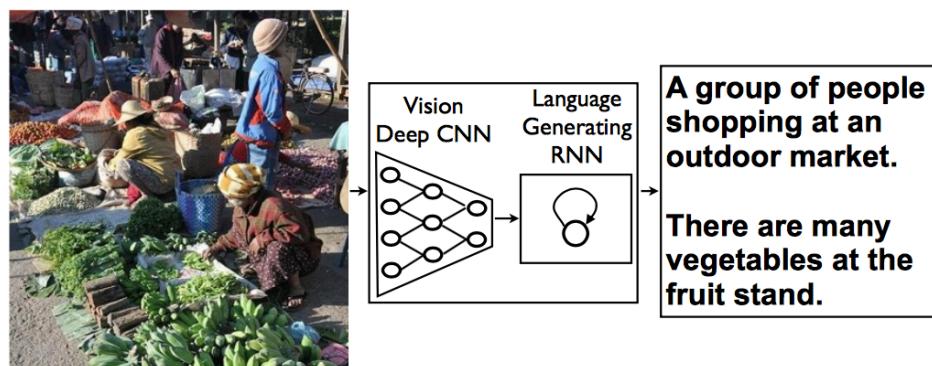
myscarlet@sina.com

目 录

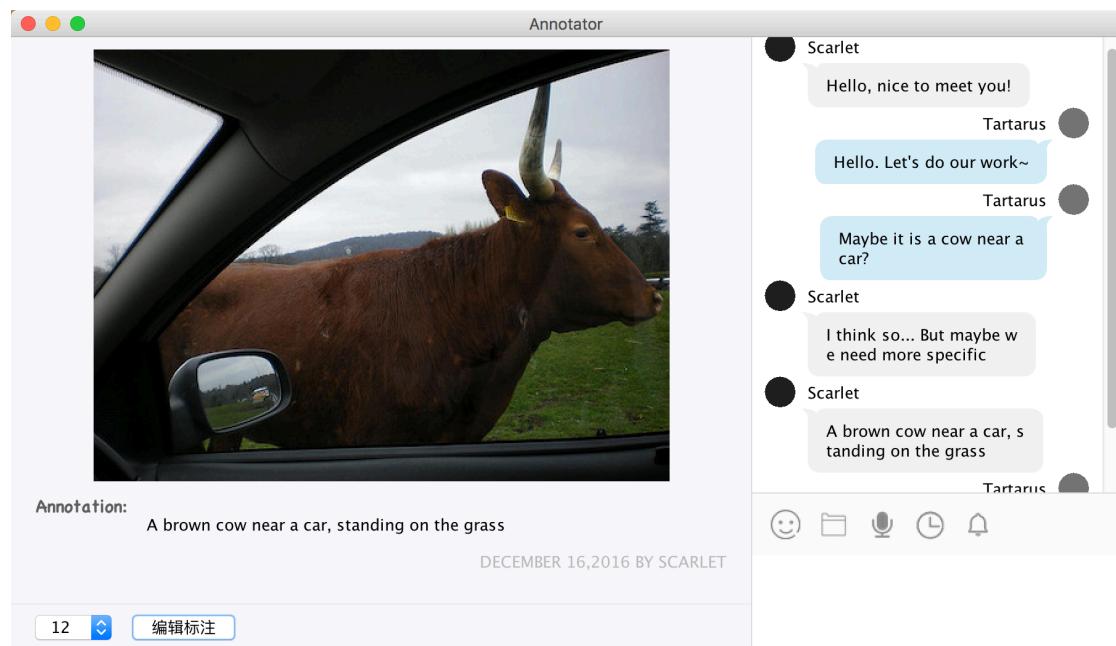
1 引言.....	1
1.1 设计目的.....	2
1.2 设计说明.....	2
2 总体设计.....	3
2.1 功能模块设计.....	3
2.2 流程图设计.....	4
3 详细设计.....	5
3.1 服务端设计.....	5
3.1.1 服务端主程序.....	5
3.1.2 数据库管理辅助程序.....	8
3.2 客户端登录窗口设计.....	9
3.3 客户端主窗口图片标注部分设计.....	11
3.4 客户端主界面聊天框部分设计.....	12
3.5 客户端历史记录窗口设计.....	16
3.6 其他工具类和函数设计.....	18
4 测试与运行.....	19
4.1 程序测试.....	19
4.2 程序运行.....	19
4.2.1 用户登录界面.....	19
4.2.2 图片显示、标注和标注界面.....	21
4.2.3 聊天界面.....	23
4.2.4 聊天记录界面.....	26
4.2.5 多人协作展示.....	27
5 总结.....	29
参考资料.....	30

1 引言

本次开发的是一款图片协同标注平台。在计算机视觉领域，有一些特殊的任务，比如 image caption，它需要机器能够给予一张输入的图片进行描述，如下图所示：



对于这种任务，往往用到的是机器学习 / 深度学习的相关知识，因此需要大量的已标注的图片和描述数据。本次开发的协同标注平台正是为了给所提供的原始图片数据让标注者有个较好的平台进行标注。与一般实验室中所用的简单的标注软件不同的，本标注平台还提供聊天窗口，并且能够处理多个标注者同时标注一张图片，大大提高了图片标注的质量，主界面截图如下：



1.1 设计目的

由上所述，针对于计算机视觉的图片描述这个任务，该平台主要是为了一些研究者提供大量优质的数据集，为了使得标注质量的提高，支持标注者进行协同合作，并用内置的聊天窗口进行讨论，大致需要由以下几个指标：

- (1) 能实现最基础的图片放映，包括图片的选择和显示。图片的放映主要在客户端上，而图片的存储则在服务端。
- (2) 能实现图片的标注功能，并实时将标注同步到服务器的数据库中。
- (3) 能实现并发标注，即多个用户可以同时标注一张图片。
- (4) 能实现聊天窗口，达到 mac 版 QQ 聊天的界面体验。
- (5) 能实现针对某一张图片会有一个单独的聊天窗口，对于不同图片，聊天窗口和内容都会发生变化。
- (6) 能实现聊天记录的实时查询，包括记录的发言人、日期等。
- (7) 能实现聊天记录存储在服务器中，并且根据每张图片的 id 不同存储不同的聊天记录。
- (8) 用户的注册和登录机制。

1.2 设计说明

本程序采用 Java 程序设计语言，遵从 google 的 java 编码规范，在 IntelliJ Idea 平台下编辑、编译与调试。在 Mac OS Sierra 10.12.1 下测试成功，本程序从算法到编写均由个人单独完成。

主要用到了 Socket 网络编程，Swing 进行 GUI 开发，Sqlite 进行数据库存储，java 中的 thread 进行并发编程。未使用其他第三方库。

2 总体设计

2.1 功能模块设计

本程序需实现的主要功能有：

- (1) 图片放映，包括图片的选择和显示。图片的放映主要在客户端上，而图片的存储则在服务端。
- (2) 图片的标注功能，并实时将标注同步到服务器的数据库中。
- (3) 并发标注，即多个用户可以同时标注一张图片。
- (4) 聊天窗口，达到 mac 版 QQ 聊天的界面体验。
- (5) 针对某一张图片会有一个单独的聊天窗口，对于不同图片，聊天窗口和内容都会发生变化。
- (6) 聊天记录的实时查询，包括记录的发言人、日期等。
- (7) 聊天记录存储在服务器中，并且根据每张图片的 id 不同存储不同的聊天记录。
- (8) 用户的注册和登录机制。

客户端程序的总体功能如图 1 所示：

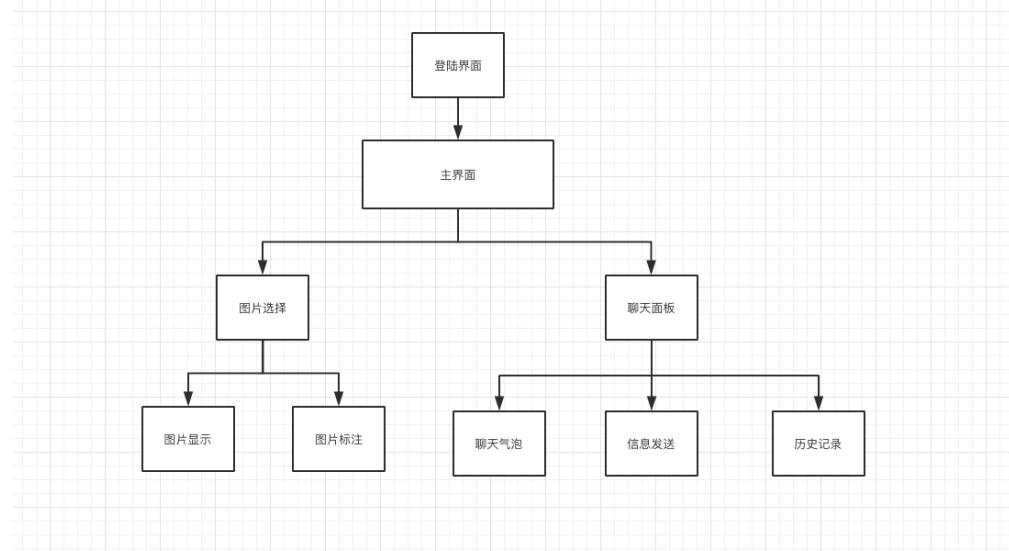


图 1 客户端总体功能

2.2 流程图设计

客户端总体流程如图 2 所示：

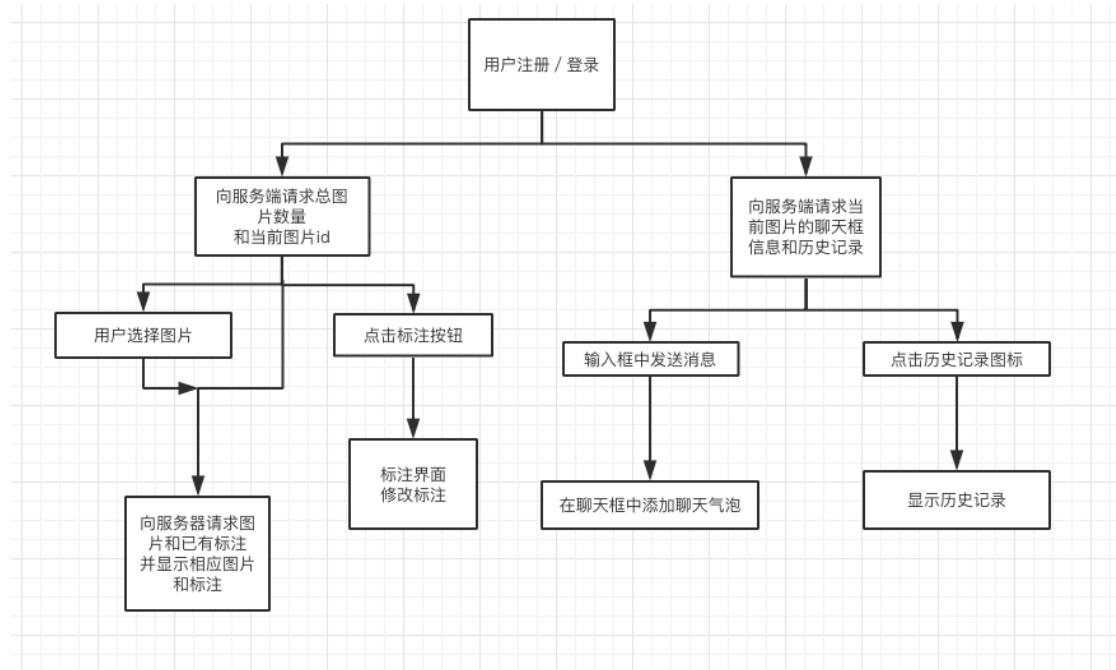


图 2 客户端总体流程图

服务端总体流程如图 3 所示：

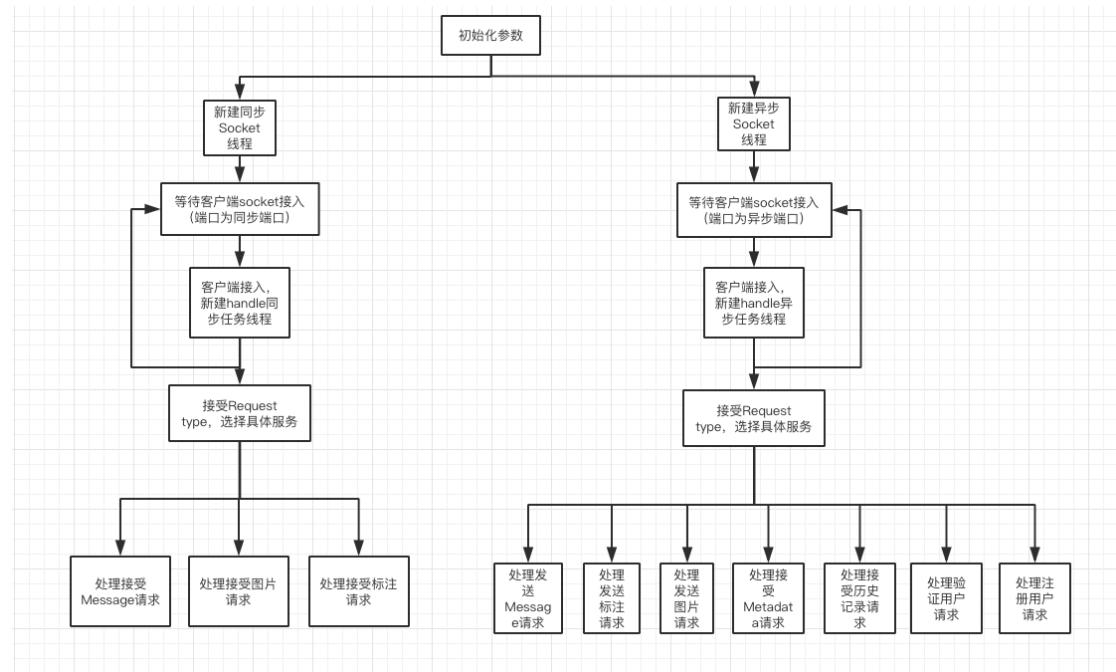


图 3 服务端总体流程图

3 详细设计

3.1 服务端设计

3.1.1 服务端主程序

Server 在创建的时候首先会初始化一些静态的变量，比如 serverExeTime 用来表示这是第几次运行服务端程序，这在后面的历史记录显示中会起到关键作用。然后创建两个服务端 socket，再同步创建两个子线程，一个线程用来循环接受同步端口的客户端 socket，另一个线程用来循环接受异步端口的客户端 socket。至于为什么要用到两类端口，是因为异步端口往往发出的请求只会执行一次，而同步端口发出的请求往往会轮询刷新，因此将这两种不同的 socket 的分离开来将有利于整体的 socket 编程。

每个线程在接受到客户端之后都会创建相应的 handleTask 线程，用来处理各式各样不同的请求，具体的解释如下所示：

1. 处理异步的请求：HandleAsynchronousTask

异步请求因为只需要运行一次，所以在请求结束之后，这个线程会自动关闭，而在代码编写的过程中，也无需加上 while 循环，如下是 HandleAsynchronousTask 类的 UML 类图：

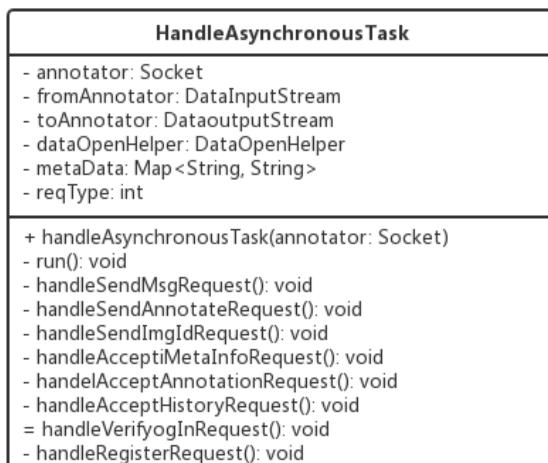


图 4 HandleAsynchronousTask 类图

这个类 implement 了 Runnable 接口，因此有个 run()函数，在 run 函数中主要采用了一个 switch 语句对请求进行选择：

```
123  switch (reqType) {  
124      case SEND_MSG_REQ:  
125          handleSendMsgRequest();  
126          break;  
127      case SEND_IMG_REQ:  
128          break;  
129      case SEND_ANNOTATE_REQ:  
130          handleSendAnnotateRequest();  
131          break;  
132      case SEND_IMG_ID_REQ:  
133          handleSendImgIdRequest();  
134          break;  
135      case ACCEPT_ANNOTATE_REQ:  
136          handleAcceptAnnotationRequest();  
137          break;  
138      case ACCEPT_META_INFO:  
139          handleAcceptMetaInfoRequest();  
140          break;  
141      case ACCEPT_HISTORY_REQ:  
142          handleAcceptHistoryRequest();  
143      case VERIFY_LOGIN_REQ:  
144          handleVerifyLogInRequest();  
145          break;  
146      case REGISTER_REQ:  
147          handleRegisterRequest();  
148          break;  
149  }
```

然后就是每一个请求各自对于输入输出流的处理函数了。

- (1) *handleSendMsgRequest*: 该函数主要处理来自客户端的发送聊天信息请求，收到信息以后，会将该信息的图片 id、作者、时间、内容都分别存入相应的数据库中。
- (2) *handleSendAnnotateRequest*: 该函数主要处理来自客户端发送标注信息请求，收到相应的标注信息以后，会将该标注的图片 id、作者、时间、标注内容都存入相应的数据库中
- (3) *handleSendImgIdRequest*: 该函数主要处理来自客户端发送图片 Id 的请求，主要功能是使得每个客户端都能共享各自浏览的图片 id，也就是每个客户端只会有一个图片 id。
- (4) *handleAcceptMetaInfoRequest*: 该函数主要处理来自客户端需要接收一些元数据的请求，比如图片的总数、现在客户端正在访问的图片 id
- (5) *handleAcceptAnnotationRequest*: 该函数主要处理来自客户端需要接收标注信息的请求，客户端先发一个图片 id 给服务端，服务端再从数据库里找到该图片现在对应的标注内容、标注作者、标注时间。
- (6) *handleAcceptHistoryRequest*: 该函数主要处理来自客户端需要接收聊天的历史记录信息的请求，通过图片 id 来获取历史用户所发的聊天记录。

(7) *handleVerifyLogInRequest*: 该函数主要处理来自客户端的验证登录信息请求，客户端会传递过来用户名和输入的密码，然后服务端从数据库中获取记录进行加密比对后传给客户端一个 Boolean 的验证信息。

(8) *handleRegisterRequest*: 该函数主要处理来自客户端的注册用户请求，收到客户端中输入的用户名和密码之后，加密存储到相应的数据库中。

2. 处理同步轮询的请求: HandleSynchronousTask

同步请求主要是为了一些客户端的刷新线程服务，比如要实时刷新聊天框中的聊天内容，而为了协同工作需要实时刷新现在用户正在访问拿个图片。

HandleSynchronousTask 的 UML 类图如下所示：

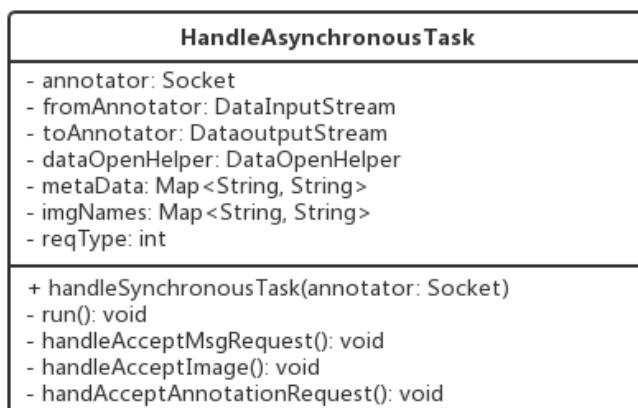


图 5 HandleSynchronousTask 类图

该同步线程和异步线程不同的是一旦服务器开始之后，客户端有几个同步的线程，服务端就会有几个同步线程，而且一直到服务器关闭才会结束，这样的好处是能够持续地接收到客户端的轮询请求。因此在代码实现上采用了一个 while 循环，如下所示：

```
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337 }  
  
while(!annotator.isClosed()) {  
    reqType = fromAnnotator.readInt();  
    switch (reqType) {  
        case ACCEPT_MSG_REG:  
            handleAcceptMsgRequest();  
            break;  
        case ACCEPT_IMG_REQ:  
            handleAcceptImage();  
            break;  
        case ACCEPT_ANNOTATE_REQ:  
            handleAcceptAnnotationRequest();  
            break;  
    }  
}
```

然后就是每一个请求各自对于输入输出流的处理函数了。

(1) *handleAcceptMsgRequest*: 该函数主要处理来自客户端的接收聊天信息请求，收到相应的图片 id 后，会将这个图片 id 在这一次的用户聊天内容传给客户端，包括内容、作者、时间等，以保证消息的实时更新。

(2) *handleAcceptImage*: 该函数主要处理来自客户端接收图片的请求，获取数据库中存储的当前浏览图片 id 之后，将服务端的相关图片文件传输给客户端。

(3) *handleAcceptAnnotationRequest*: 该函数主要处理来自客户端接收标注信息请求，因为每一个图片的标注信息会在协同工作的时候时刻发生变化，所以需要一个同步线程来处理这个协同关系。

3.1.2 数据库管理辅助程序

当然服务器线程本身是不具备处理数据库的功能，而要把旧的数据保留下 来必须通过数据库的方式，因此在此就设计了一个 DataOpenHelper 类专门用来对数据库的读出和写入进行处理。

首先在数据库中定义三个表格，分别是：

- (1) annotation : 用于存储标注者的标注信息，以后可以作为研究者的数据集使用。
- (2) chat_record : 用于存储用户的聊天内容，主要是为了聊天气泡的显示和聊天记录的显示。
- (3) user : 用于存储用户的登录信息。

表格创建的 SQL 语言如下：

```
12  private final String createAnnotationTableSql =
13      "CREATE TABLE IF NOT EXISTS annotation" +
14          "(ImgId TEXT PRIMARY KEY    NOT NULL," +
15              " Description TEXT          NOT NULL," +
16                  " Author TEXT,             " +
17                      " Date TEXT              " +
18                  ");";
19  private final String createChatRecordSql =
20      "CREATE TABLE IF NOT EXISTS chat_record" +
21          "(ImgId INT    NOT NULL," +
22              " ChatId INT      NOT NULL," +
23                  " Author TEXT     NOT NULL," +
24                      " ChatContent TEXT  NOT NULL," +
25                          " Date TEXT        " +
26                      ");";
27  private final String createUserRecordSql =
28      "CREATE TABLE IF NOT EXISTS user" +
29          "(user_id TEXT PRIMARY KEY  NOT NULL," +
30              " password TEXT          NOT NULL," +
31                  " register_time TEXT     NOT NULL" +
32                  ");";
```

除了这三张表格以外，还以 Map 的形式存了一些 Metadata，比如图片的总数、现在访问的图片 id 等。

数据库采用的是轻量级数据库 SQLite，直接导入 sqlite-jdbc.jar 包之后就可以正常使用了，初始化数据库后，会在相应的目录下创建一个.db 文件存储相应的数据库数据。

DataOpenHelper 主要是提供一些操作 SQLite 数据库的方法，其 UML 类图如下所示：

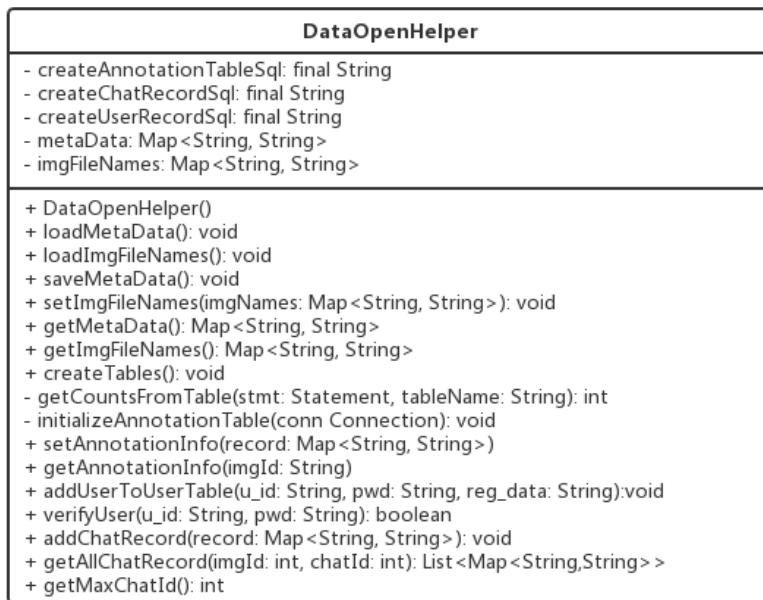


图 6 DataOpenHelper 类图

由于这个类是用来操控数据库的，功能大致都在 UML 类图里了，而用到的也是一些 SQL 语言的语法，因此在这里就不一一赘述了。

3.2 客户端登录窗口设计

由在进入主窗口程序之前，需要经过一个用户验证界面，用于用户的注册和登录。这个窗口的布局也很简单，用了一个 BorderLayout，上面显示一个背景图片，下边显示输入栏和“登录”、“注册”的按钮。由下面的 LogInWindow 的 UML 类图可以看出该界面有多少相应的 widgets：

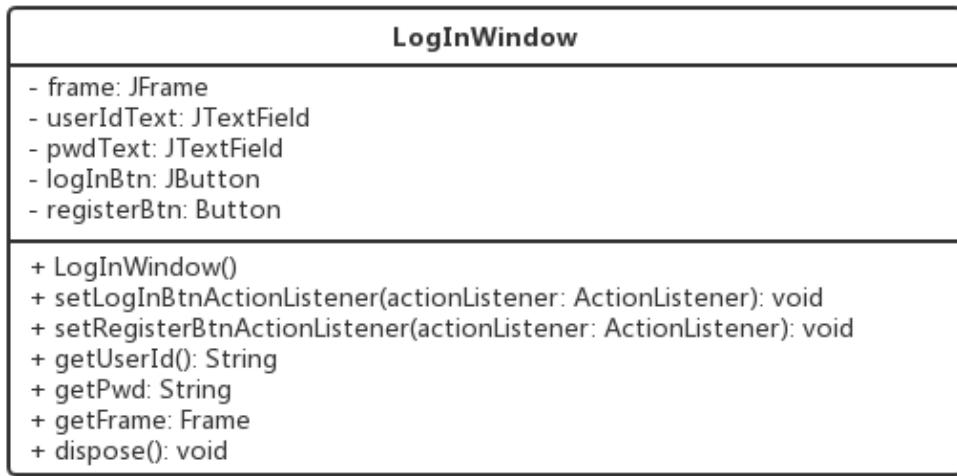


图 7 LogInWindow 类图

提供外部的接口，可以在客户端主程序中调用以下几个方法：

- (1) *setLogInBtnActionListener*: 将 login 按钮对外添加一个 Action Listener，可以在外部的方法中对其进行调用，然后执行外部定义的一个 ActionPerformed。
- (2) *setRegisterBtnActionListener*: 将 register 按钮对外添加一个 Action Listener，可以在外部的方法中对其进行调用，然后执行外部定义的一个 ActionPerfomed
- (3) *getUserId*: 拿到输入框中的用户名
- (4) *getPwd*: 拿到输入框中的密码
- (5) *getFrame*: 取得 logInWindow 的主 frame
- (6) *dispose*: 将 LogInWindow 的 frame 给取消掉。

而验证逻辑是每次用户输入用户名密码之后，如果点击登陆按钮，客户端就会向服务端发起一个请求，并传输相应的用户输入，最后服务端验证完毕后会传过来一个 Boolean 值，通过这个 Boolean 值来判断用户名密码验证是否成功。如果成功的话，就 dispose 掉 LogInWindow 并且打开主界面。注册也是同理，首先会弹出一个注册框，用户在里面输入用户名密码之后会将其传输到服务端，服务端相应地加密存在数据库中。

3.3 客户端主窗口图片标注部分设计

本次所设计的系统最为核心的第一部分就是图片显示和图片标注了。首先在主窗口左半部分专门作为图片显示和标注窗口，在源代码里主要用 AnnotationClient 类里的 initImagePanel() 来实现该部分画板的绘制。它的布局大

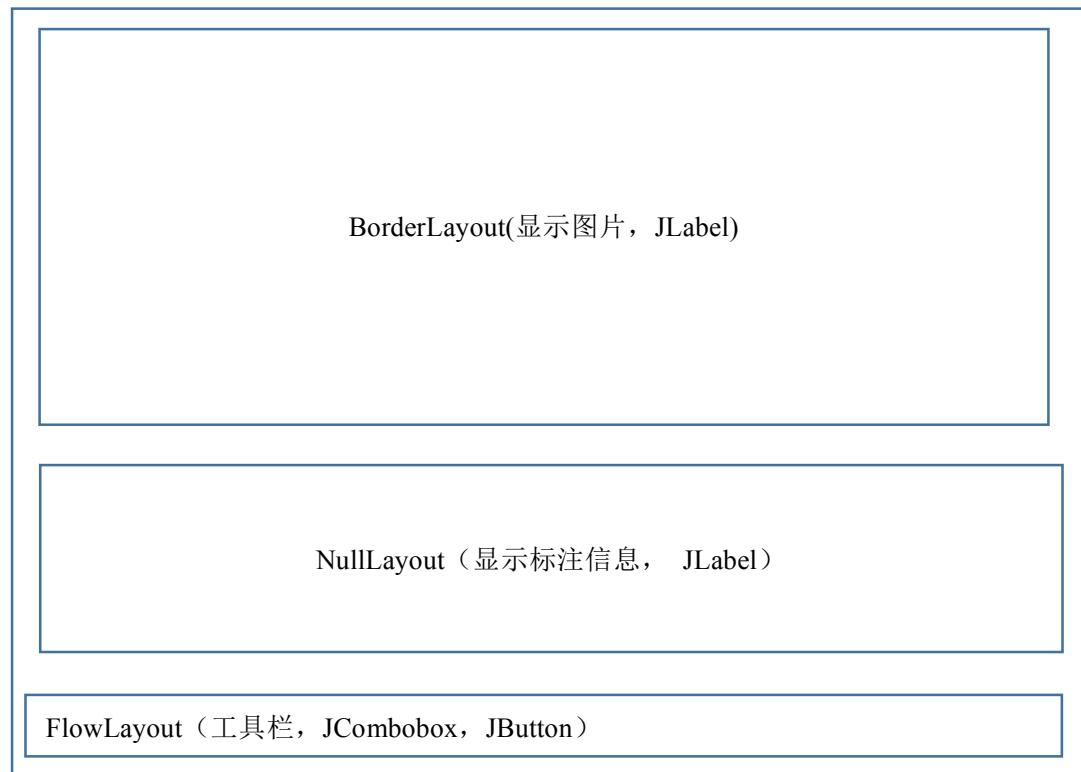


图 8 ImagePanel 总体布局

致是这样的：

当点击工具栏中的“修改标注”的按钮时，会新建一个 AnnotateWindow，这是一个新的标注界面，因此用一个新的类来进行表示，下图是其 UML 图：

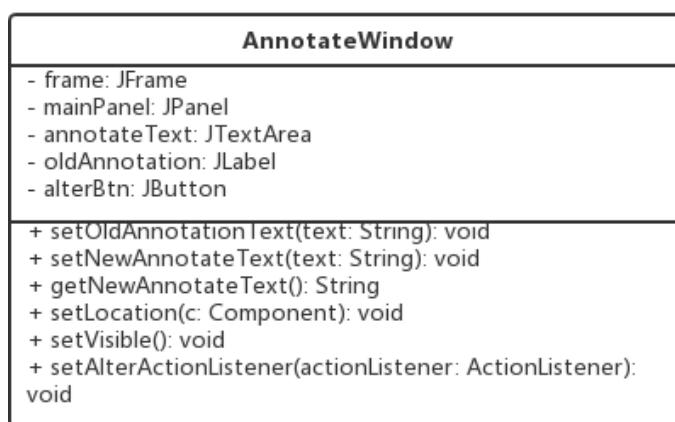


图 9 AnnotateWindow 类图

- (1) `setOldAnnotationText`: 在类的外面调用该方法, 改变显示在标注窗口中的“原”标注信息。
- (2) `setNewAnnotateText`: 在类的外面调用该方法, 改变现实在标注窗口中的“新”标注信息
- (3) `getNewAnnotateText`: 在类的外面调用该方法, 可以获得新的标注信息。
- (4) `setLocation`: 为了使得标注窗口在主窗口的中央。
- (5) `setVisible`: 在类的外面是的标注窗口出现。
- (6) `setAlterActionListener`: 在类的外面新建对于“修改”按钮的 `ActionListener`。即可在类外定义 `ActionPerformed` 与服务器进行交互。

除了界面设计以外, 图片的显示和标注需要一定的逻辑与服务器进行正常交互, 交互的过程如下流程图所示:

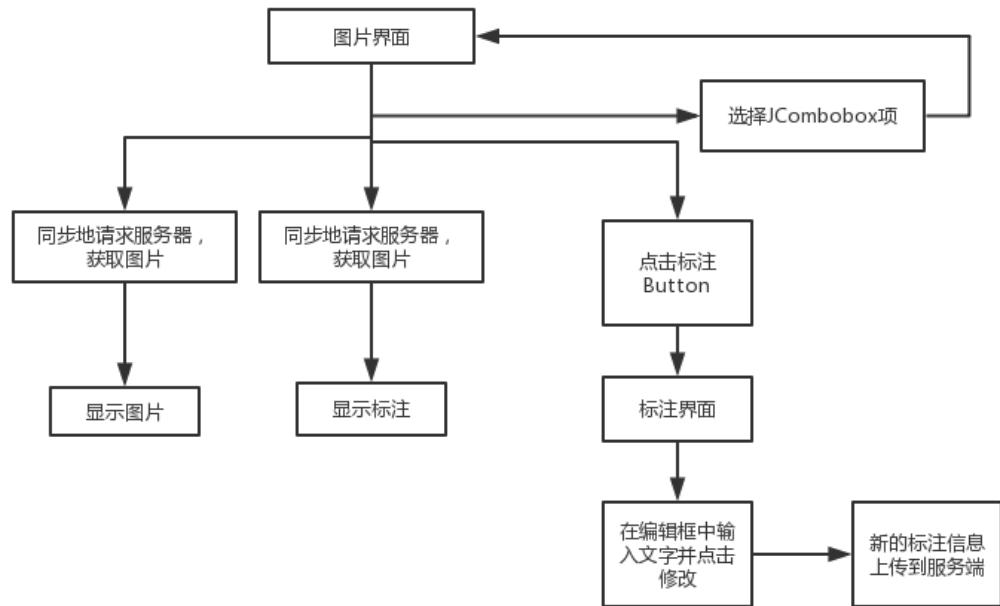


图 10 图片界面与服务器交互流程图

3.4 客户端主界面聊天框部分设计

本系统的另一个核心的部分是聊天系统, 聊天界面的外观跟一些主流的社交软件类似, 分成上中下三个部分, 上方是聊天气泡区域, 中间是工具栏, 下方是

用来提供用户输入使用，在源代码里主要用 AnnotationClient 类里的 initChatPanel() 来实现该部分画板的绘制，该界面布局和初始化布局如下所示：

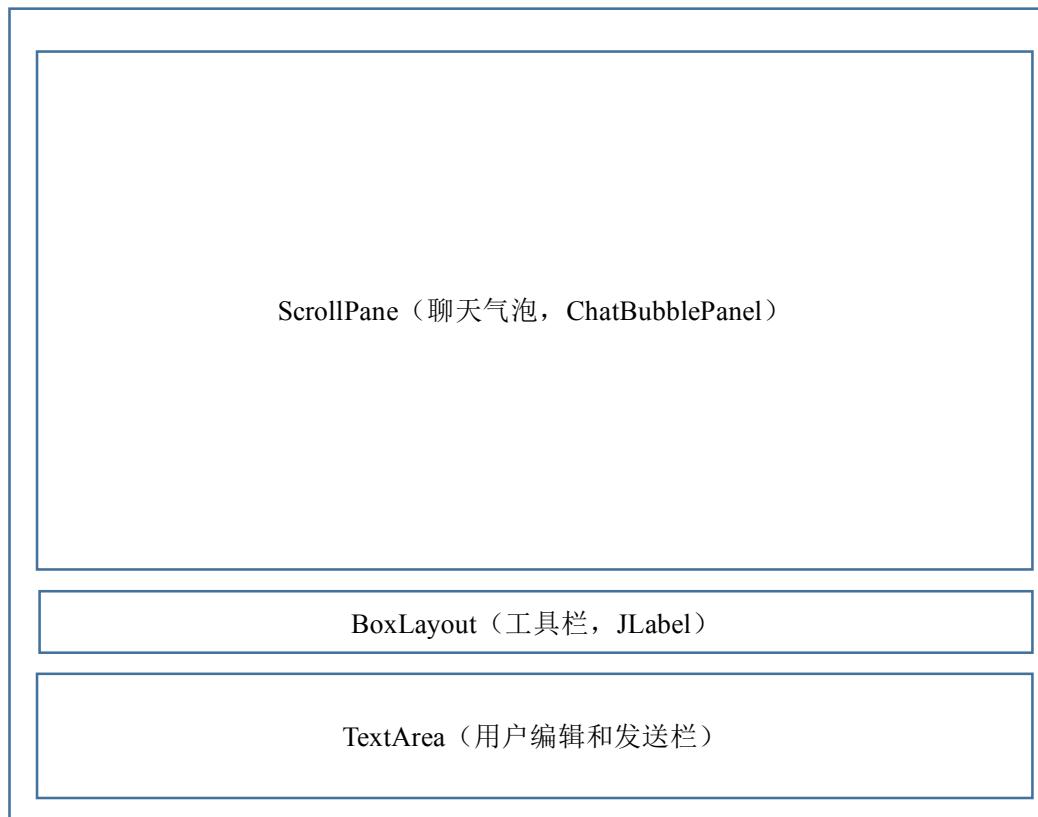


图 11 ChatPanel 布局

其中如果想要绘制出一个好看的用户聊天界面，需要自定一个 ChatBubblePanel，其功能是显示用户头像、用户名，并以气泡作为容器放置聊天内容，该气泡会随着聊天内容的长短进行变换大小、换行等，并且能够识别聊天信息是来自他人还是自己分别变更聊天气泡的颜色和左右位置。ChatBubblePanel 的 UML 类图如下所示：

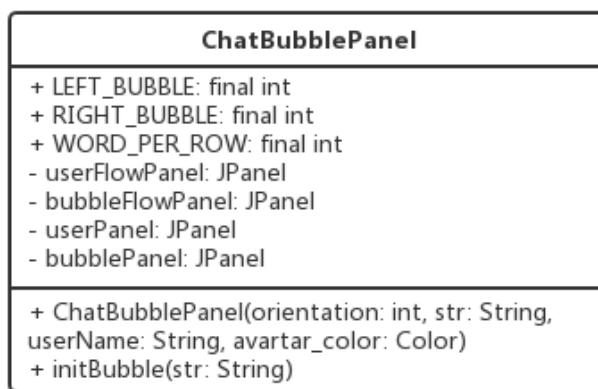


图 12 ChatBubblePanel

Panel，其中 userPanel 在实例化过程中使用了自定义的一个 JPanel 子类 avatarPanel，其功能是在一个 FlowLayout 上布局一个有色实心圆作为用户头像（重写 paintComponent），然后在添加一个 JLabel 作为用户名文字的容器，具体实现如下面代码所示：

```
125  /**
126  * avatar Panel for placing a circle for avatar and userName
127  */
128  public static class avatarPanel extends JPanel {
129      private JLabel userName;
130      int orientation;
131      Color fill_color;
132      public avatarPanel(int orientation, String u_name, Color fill) {
133          this.orientation = orientation;
134          if (orientation == LEFT_BUBBLE)
135              this.setLayout(new FlowLayout(FlowLayout.LEFT));
136          else
137              this.setLayout(new FlowLayout(FlowLayout.RIGHT));
138          this.userName = new JLabel();
139          this.userName.setText(u_name);
140          this.add(this.userName);
141          this.setBorder(new EmptyBorder(0, 30, 0, 30));
142          this.fill_color = fill;
143      }
144
145      @Override
146      public void paintComponent(final Graphics g) {
147          Graphics2D g2d = (Graphics2D)g;
148          // Assume x, y, and diameter are instance variables.
149          Ellipse2D.Double circle;
150          if(orientation == LEFT_BUBBLE)
151              circle = new Ellipse2D.Double(0, 0, 25, 25);
152          else
153              circle = new Ellipse2D.Double(getWidth() - 26, 0, 25, 25);
154          g2d.setColor(fill_color);
155          g2d.fill(circle);
156      }
157  }
```

可以看出这个 Panel 会随着传入的方向 (orientation) 的不同将头像、用户名放置在不同的位置。

另外，ChatBubblePanel 中所用到的 bubbleFlowLayout 作为一个 JPanel 的对象实例化过程中会进行判断，如果是左侧的，就用继承于 JPanel 类的 LeftArrowBubble 进行实例化，否则就用继承于 JPanel 类的 RightArrowBubble 进行实例化。我们截取 LeftArrowBubble 为例：

```
128  /**
129  * Left bubble with an arrow
130  */
131  public static class LeftArrowBubble extends JPanel {
132      public static int grey_bubble = 0xF3F3F3;
133      public LeftArrowBubble() {
134          this.setBorder(new EmptyBorder(10, 20, 10, 15));
135          this.setLayout(new GridLayout(0, 1));
136      }
137  
```

```

137
138
139 @Override
140     protected void paintComponent(final Graphics g) {
141         final Graphics2D graphics2D = (Graphics2D) g;
142         RenderingHints qualityHints = new RenderingHints(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
143         qualityHints.put(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);
144         graphics2D.setRenderingHints(qualityHints);
145         graphics2D.setPaint(new Color(grey_bubble));
146         int width = getWidth();
147         int height = getHeight();
148         GeneralPath path = new GeneralPath();
149         path.moveTo(5, 10);
150         path.curveTo(5, 10, 7, 5, 0, 0);
151         path.curveTo(0, 0, 12, 0, 12, 5);
152         path.curveTo(12, 5, 12, 0, 20, 0);
153         path.lineTo(width - 10, 0);
154         path.curveTo(width - 10, 0, width, 0, width, 10);
155         path.lineTo(width, height - 10);
156         path.curveTo(width, height - 10, width, height, width - 10, height);
157         path.lineTo(15, height);
158         path.curveTo(15, height, 5, height, 5, height - 10);
159         path.lineTo(5, 15);
160         path.closePath();
161         graphics2D.fill(path);
162     }

```

另外可以看出在绘制气泡的过程中，采用了 GeneralPath 进行精确绘制，并且与气泡内容的 width 和 height 相关联，从而做到对于不同长度的聊天内容能够实现不同大小的气泡。

在介绍完聊天气泡之后就是工具栏了。工具栏的实现并没有什么特殊的地方，采用了一个 BoxLayout 将一些工具图标以 JLabel 的方式进行呈现。并且设置了 MouseListener，当鼠标移到相应的工具图标的时候，该图标就会从灰色变成蓝色，增加了与用户之间的交互。实现的过程如下所示：

```

653 smileLabel.addMouseListener(new MouseAdapter() {
654     @Override
655     public void mouseClicked(MouseEvent e) {
656         super.mouseClicked(e);
657         JOptionPane.showMessageDialog(null, "这个功能还没实现",
658             "", JOptionPane.WARNING_MESSAGE);
659     }
660
661 smileLabel.addMouseListener(new MouseAdapter() {
662     @Override
663     public void mouseEntered(MouseEvent e) {
664         try {
665             smileLabel.setIcon(new ImageIcon(ImageIO.read(new File("res/smile-hover.png"))));
666         } catch (Exception err) {
667             err.printStackTrace();
668         }
669     }
670
671     @Override
672     public void mouseExited(MouseEvent e) {
673         try {
674             smileLabel.setIcon(new ImageIcon(ImageIO.read(new File("res/smile.png"))));
675         } catch (Exception err) {
676             err.printStackTrace();
677         }
678     });
679     // Set file label hover action
680     fileLabel.addMouseListener(new MouseAdapter() {...});
681     // Set voice label hover action
682     voiceLabel.addMouseListener(new MouseAdapter() {...});
683     // Click history label, open a history window
684     historyLabel.addMouseListener(new MouseAdapter() {...});
685     // Set remind label hover action
686     remindLabel.addMouseListener(new MouseAdapter() {...});

```

可以看出只是单纯地将灰色图标换成了蓝色的图标，而实现了相应的效果。

最后一部分是发送文本框的实现，在这里实现的是一个简单的文本输入 JTextArea 并且添加一个 KeyListener。当用户按下回车键的时候，消息就会及时发送出去，并且显示在聊天框内。KeyListener 如下所示：

```
549  /**
550  * Init some listeners.
551  */
552  private void initActionListener() {
553      // When pressed "Enter" on keyboard
554      sendContent.addKeyListener((KeyAdapter) keyPressed(e) -> {
555          if(e.getKeyCode() == KeyEvent.VK_ENTER) {
556              // Start a new thread to send message to send thread.
557              e.consume();
558              (Thread) run() -> {
559                  try {
560                      synchronized (sendMsg) {
561                          sendMsg.setMsg(sendContent.getText(), author, Utils.getCurTimeText());
562                          sendMsg.notify();
563                          sendContent.setText("");
564                      }
565                  } catch (Exception e) {
566                      e.printStackTrace();
567                  }
568              }.start();
569          }
570      });
571  }
572
573
574
575 }
```

这里 sendMsg 是一个自定义的 Message 类，如下所示：

```
6  public class Message {
7      private String msg;
8      private String author;
9      private String time;
10
11     public Message(String str) { this.msg=str; }
12
13     public String getMsg() { return msg; }
14
15     public String getAuthor() { return author; }
16
17     public String getTime() { return time; }
18
19     public void setMsg(String msg, String author, String time) {...}
20
21
22
23
24
25
26
27
28
29
30
31
32
33 }
```

它可以作为一个同步变量在线程之间互相交互，将主线程的 message 发送给专门用来处理与服务器交互的 SendMessage 线程。

3.5 客户端历史记录窗口设计

作为一个具备聊天功能的标注平台，历史记录的功能也不能或缺。历史记录通过 3.4 节所讲的工具栏中的历史图标点击进入，会弹出一个 HistoryWindow，它作为一个单独的类，UML 类图如下所示：

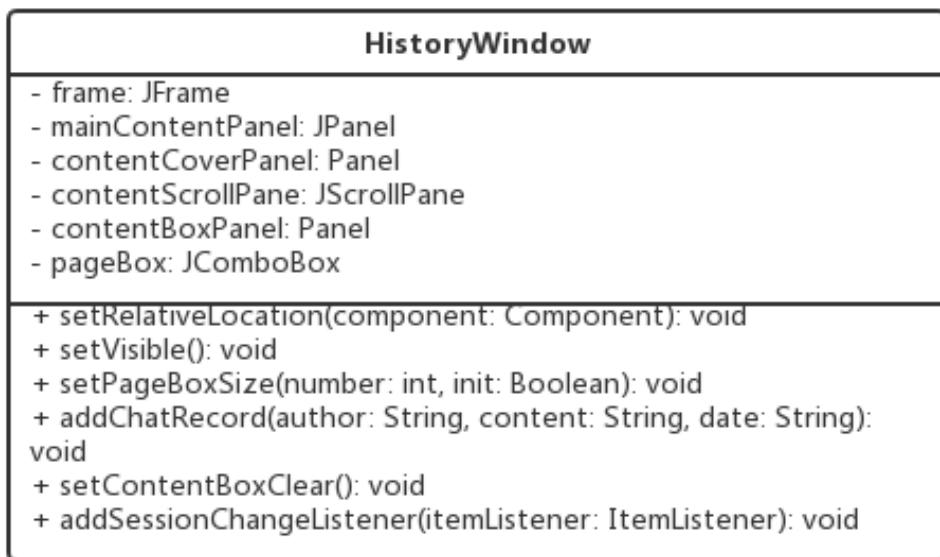


图 13 HistoryWindow 类图

- (1) *setRelativeLocation*: 在外部对该方法进行调用, 为了使得历史记录窗口在主窗口中央。
- (2) *setVisible*: 在外部对该方法进行调用, 为了使得历史记录窗口可见。
- (3) *setPageBoxSize*: 在外部对该方法进行调用, 设置 PageBox 的总页数。
- (4) *addChatRecord*: 新增一项聊天记录到历史记录聊天面板上
- (5) *setContentBoxClear*: 使得历史记录聊天面板的内容全部清空。
- (6) *addSessionChangeListener*: 在外部对 PageBox 页数更改时候的动作进行定义, 增加相应的 ItemListener。

在具体的实现过程中, 对于每一个聊天记录, 都会定义一个新的 ChatBoxPanel, 它继承 JPanel 类, 用于放置用户名、发言时间、发言内容, 该类的构造如下所示:

```
113  /*
114  * Chat Box Panel for each record
115  */
116  private class ChatBoxPanel extends JPanel {
117      private JPanel authorDateBar;
118      private JPanel contentPanel;
119      private JLabel authorLabel;
120      private JLabel dateLabel;
121      private JLabel contentLabel;
122      public ChatBoxPanel(String author, String content, String date) {
123          setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
124          authorDateBar = new JPanel(new FlowLayout(FlowLayout.LEFT));
125          contentPanel = new JPanel(new GridLayout(0, 1));
126          authorLabel = new JLabel(author);
127          dateLabel = new JLabel(date);
128          authorDateBar.setBorder(new EmptyBorder(2, 5, 0, 2));
129          authorLabel.setFont(new Font("Menlo", Font.BOLD, 12));
130          authorLabel.setForeground(new Color(0x448A35));
131          dateLabel.setFont(new Font("Menlo", Font.BOLD, 12));
132          dateLabel.setForeground(new Color(0x448A35));
133          contentLabel = new JLabel(Utils.stringAfterNewline(content, 50));
134          contentLabel.setBorder(new EmptyBorder(0, 10, 1, 2));
135          authorDateBar.add(this.authorLabel);
136          authorDateBar.add(this.dateLabel);
137          contentPanel.add(this.contentLabel);
138          add(authorDateBar);
139          add(contentPanel);
140          setBorder(new EmptyBorder(2, 0, 4, 0));
141          authorDateBar.setBackground(new Color(dark_white));
142          contentPanel.setBackground(new Color(dark_white));
143          setBackground(new Color(dark_white));
144      }
145 }
```

3.6 其他工具类和函数设计

除却上述所示的一些核心功能以外，还定义了一些工具类和函数。

1. HintTextArea、HintTextField、HintPasswordField:

这三个类主要是继承并扩展了相应的 JTextArea、JTextField 和 JPasswordField 类，在原先的基础上增加了一些提示文字，即无法编辑，在用户输入其它文字时就会消失的提示文字。

2. Utils

这是一个存放了大量静态工具函数的类，主要包含下面几个函数：

- (1) generateAnnotationRecord: 这个函数主要用于给定相关信息产生一个标注信息 Map 在服务器与客户端，服务器主程序与数据库进行互相传输文件。
- (2) generateChatRecord: 同(1)，只是将标注信息改成了聊天内容
- (3) getCurTimeText: 获得当前的时间，并且用合适的形式转换成 String
- (4) getEncryptedPwd: 将密码进行加密，在这里采用了移位加密的简单方式。

- (5) getRandomColor: 针对给定的一些颜色，随机返回这些颜色中的其中一个。
- (6) stringAfterNewline: 主要针对 JLabel 不能换行的缺点，给定一个 maxWordPerLine，如果超过这个值，就在本行的前后加上 html 中的
</br>标签用于换行。
- (7) formatAuthorDateInfo: 格式化输出标注者和标注日期，返回一个相对格式比较漂亮的 String。

4 测试与运行

4.1 程序测试

在进行不断地试错后，服务器与客户端均运行正常，基本功能都能达到预期。在做压力测试的时候，也能正常接入多个客户端进行同时操作。

4.2 程序运行

4.2.1 用户登录界面

首先进入用户登录界面如下所示：

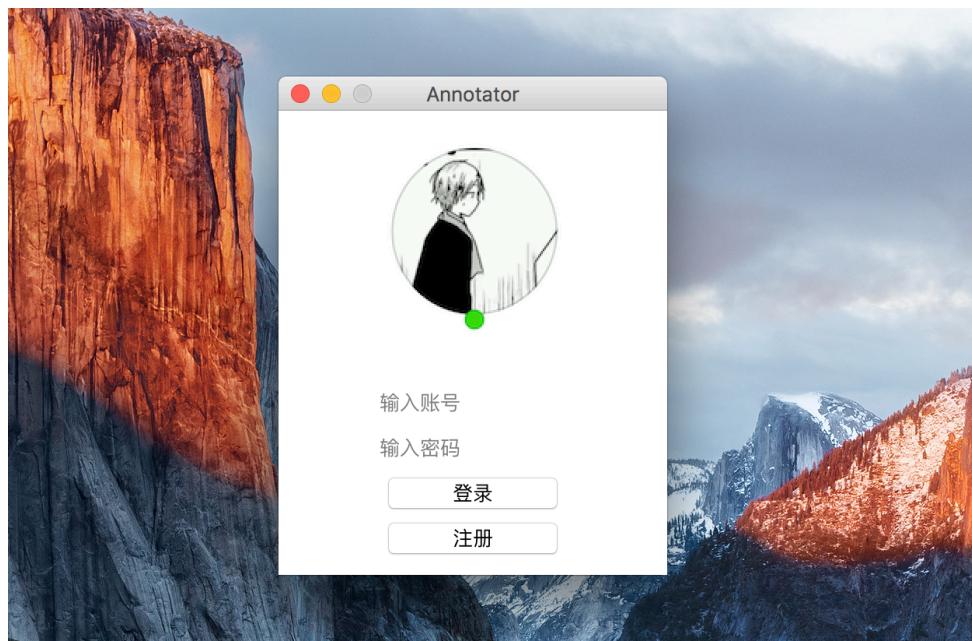


图 14 用户登录界面

然后点击注册按钮后，会弹出注册界面。



图 15 进入注册界面

注册好之后提示注册完毕。

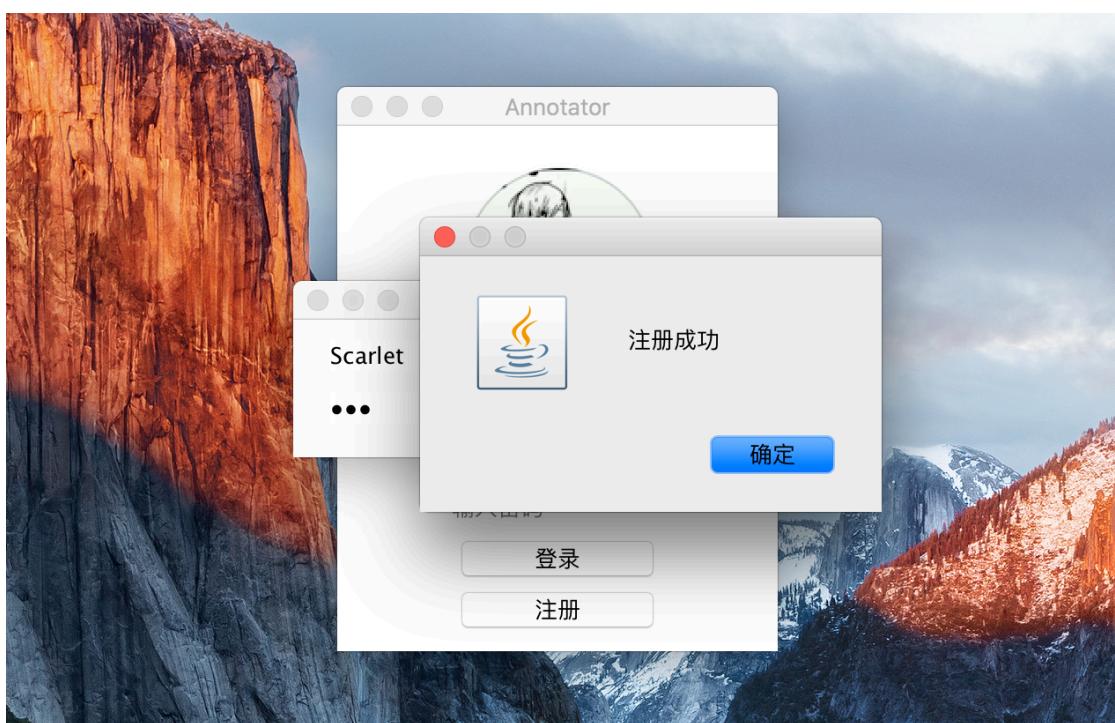


图 16 注册成功

注册好之后就可以正常登录了。

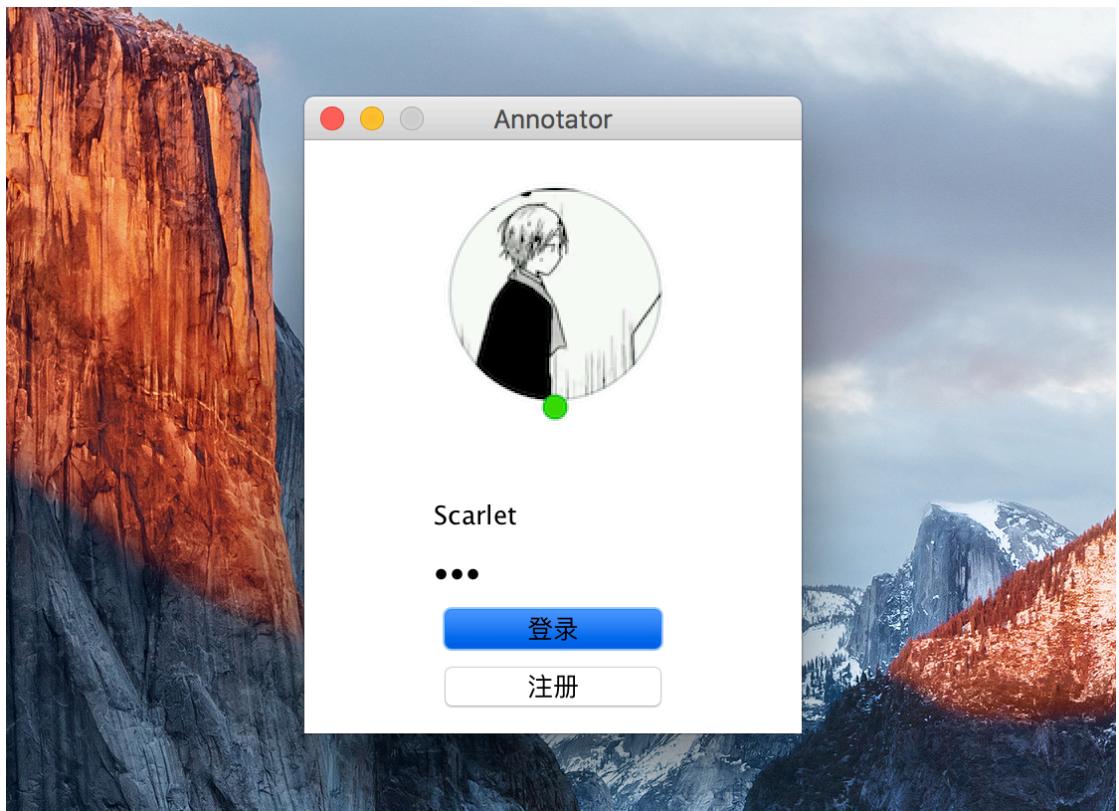


图 17 登录

4.2.2 图片显示、标注和标注界面

登录后，出现主界面，此时没有标注，也没有聊天信息：

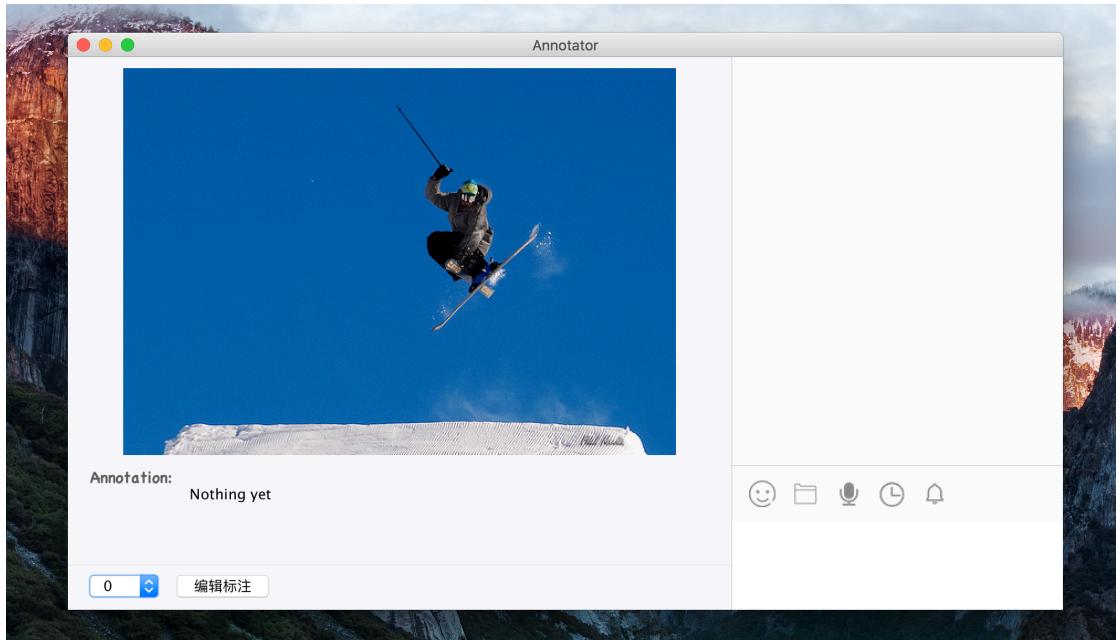


图 18 主界面

使用右侧的选择栏可以通过图片 id 选择不同的图片。

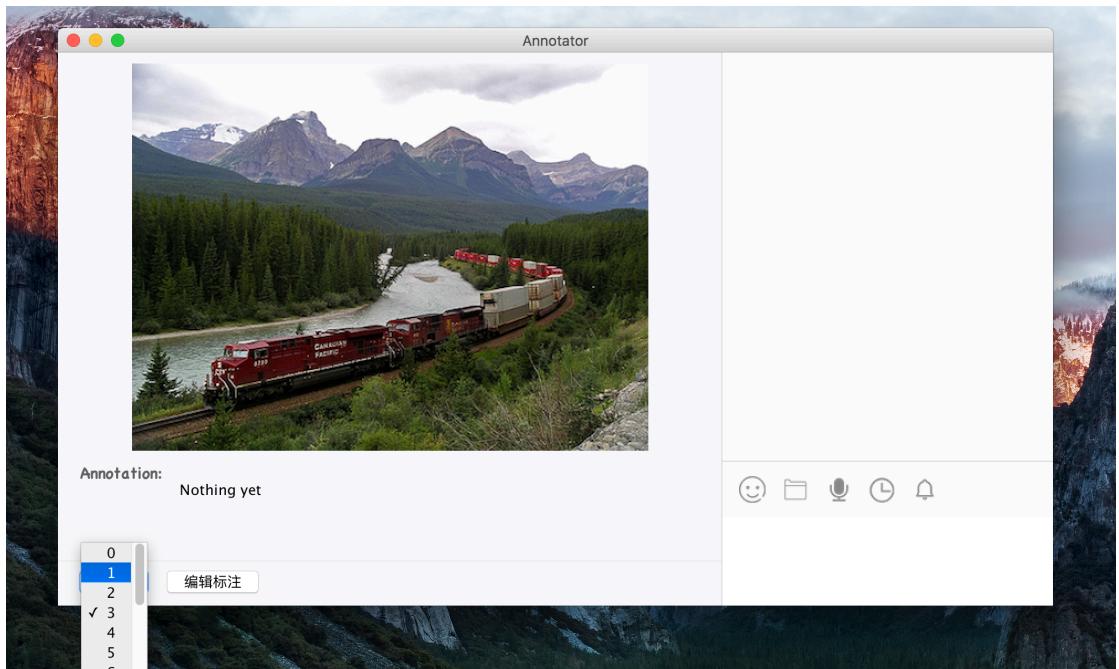


图 19 选择图片

使然后点击编辑标注，会弹出标注窗口，上面的是原标注（如果没有标注则显示 Nothing yet）

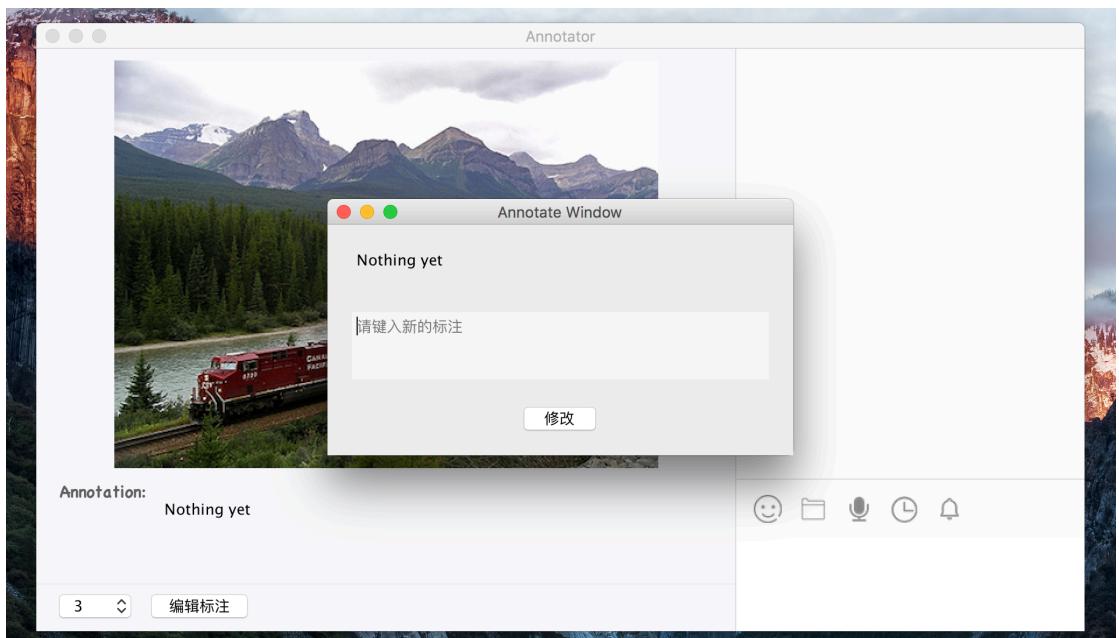


图 20 标注界面

修改完标注后，对应的标注界面和主界面的标注信息都会发生改变，并且提示修改成功。

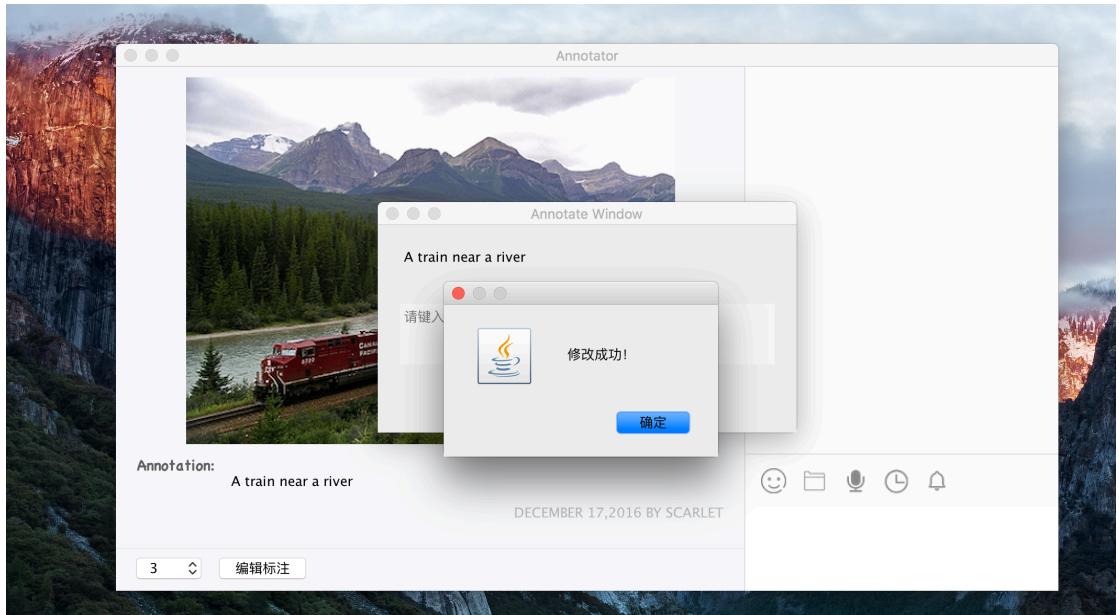


图 21 修改标注

可以看到不仅标注信息变了，还自动保存了修改时间和修改作者。

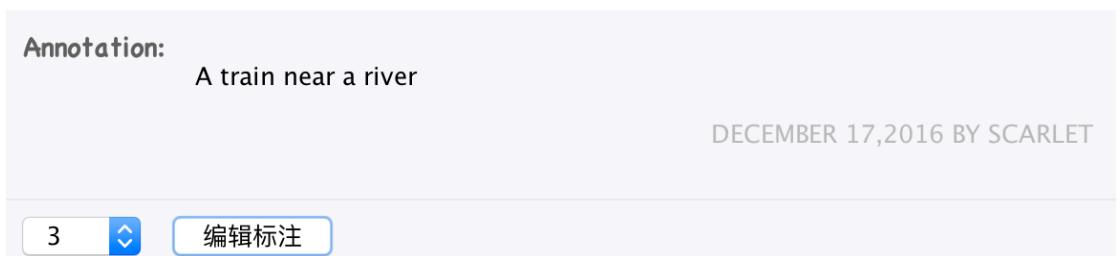


图 22 修改标注后的结果

4.2.3 聊天界面

首先在下方输入框中输入一段文字：

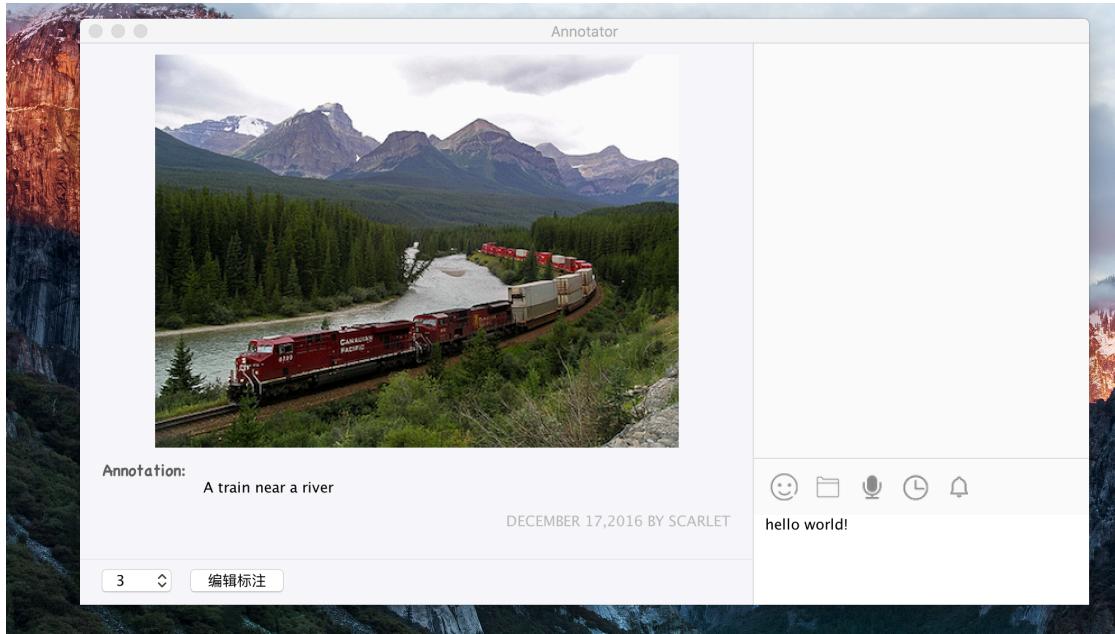


图 23 输入框输入文字

然后按下回车（enter）键，会发现上方聊天窗口中已经出现了一个合适长度的气泡：

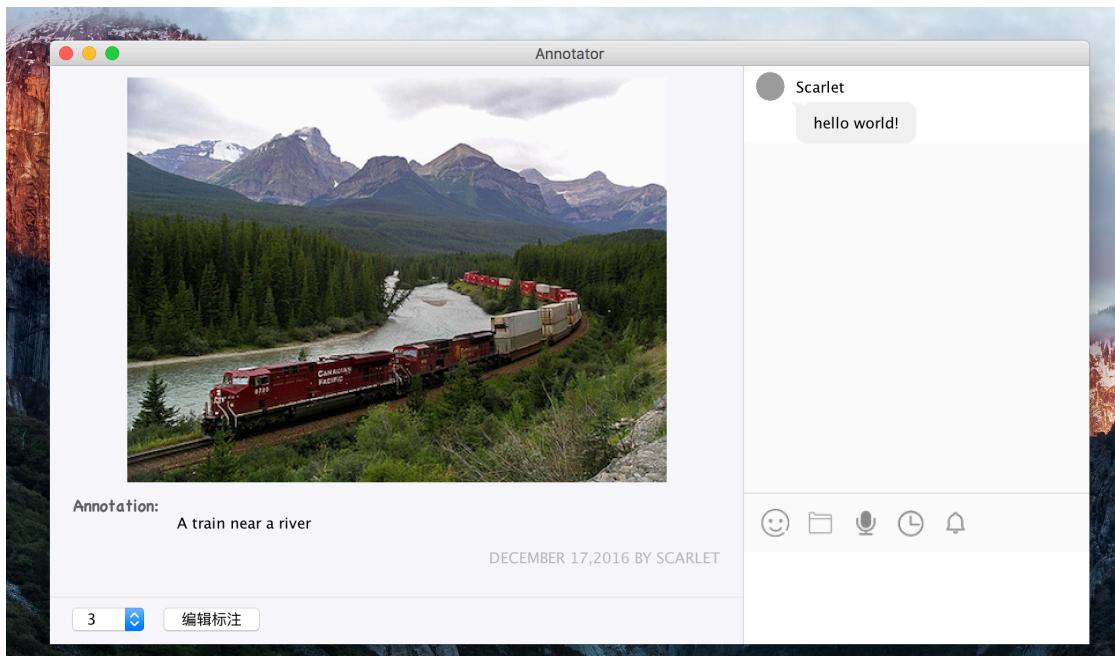


图 24 气泡显示

但当切换一张图片的时候，上方聊天窗口的气泡就会消失：也就是新启动一段对话：

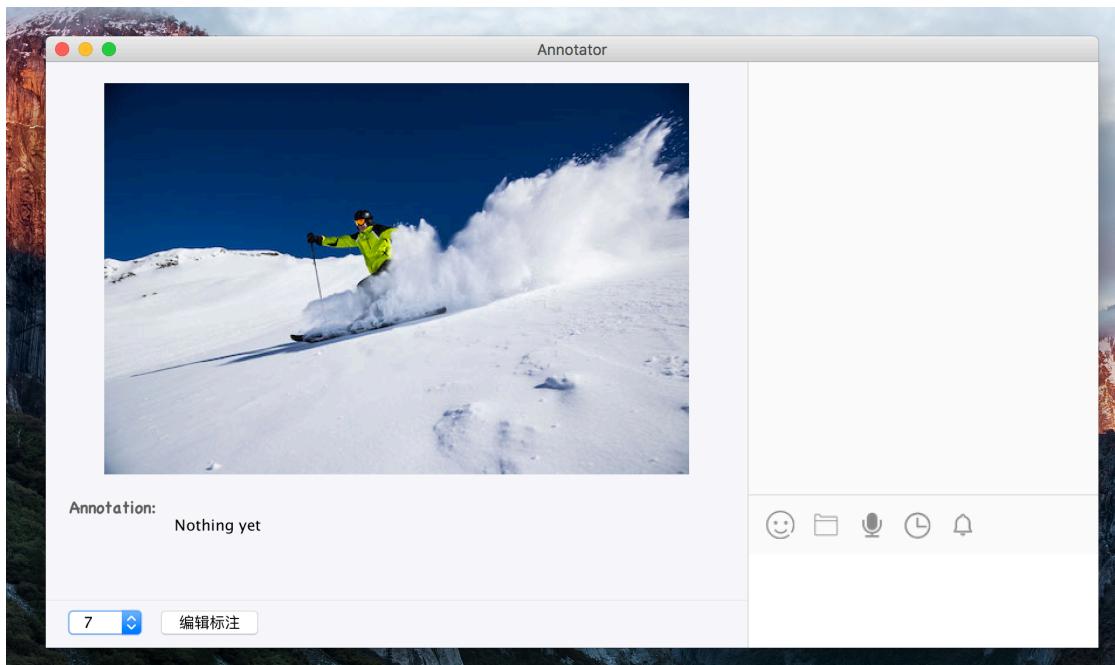


图 25 新图片新对话

另外就是一个小功能，当鼠标浮在聊天面板的图标上时，图标会显示成蓝色的图标：

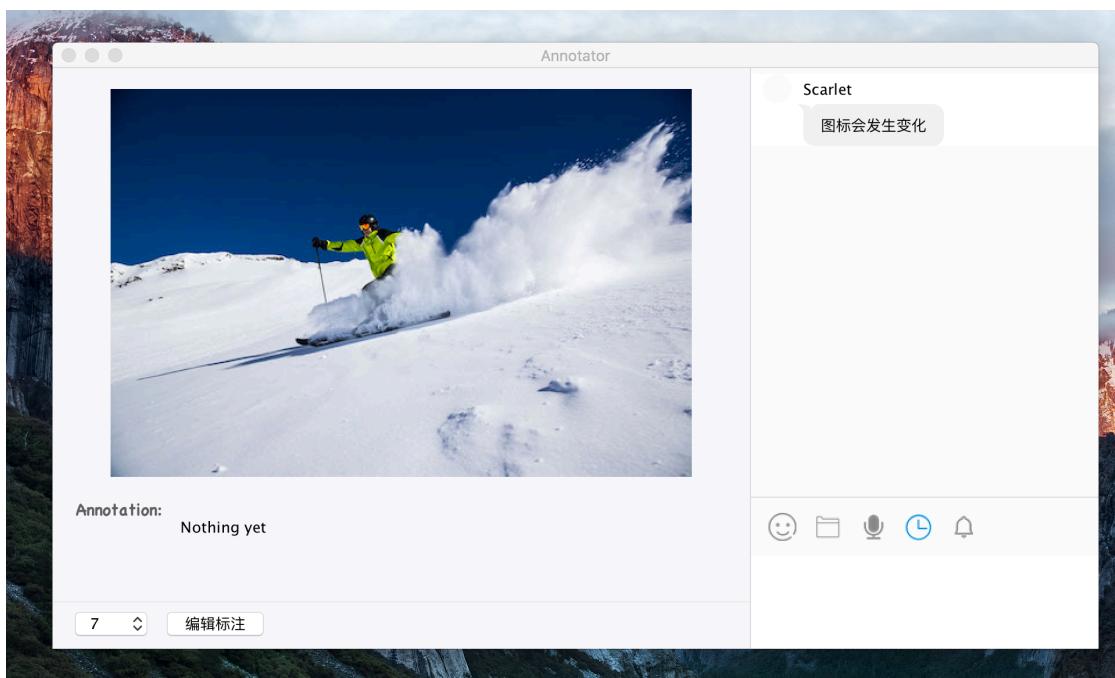


图 26 图标变色

4.2.4 聊天记录界面

聊天记录主要是通过服务器的启动次数来进行分隔，按下主界面的历史记录按钮，会出现聊天记录界面：

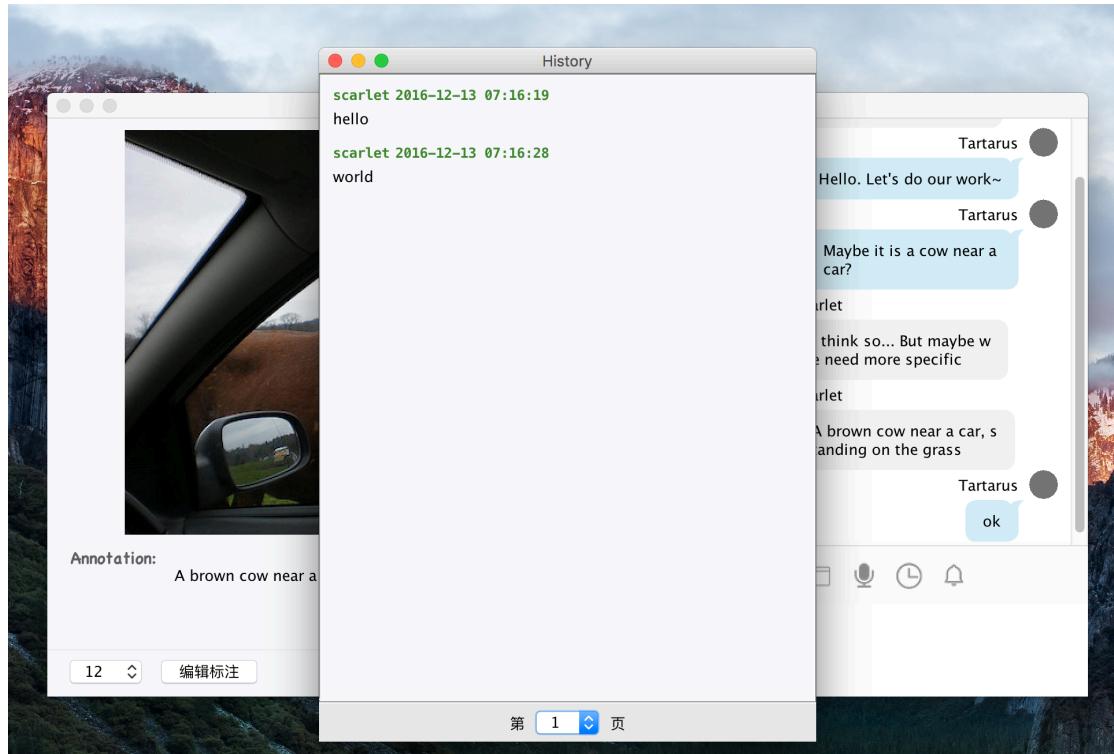


图 27 聊天记录界面 1

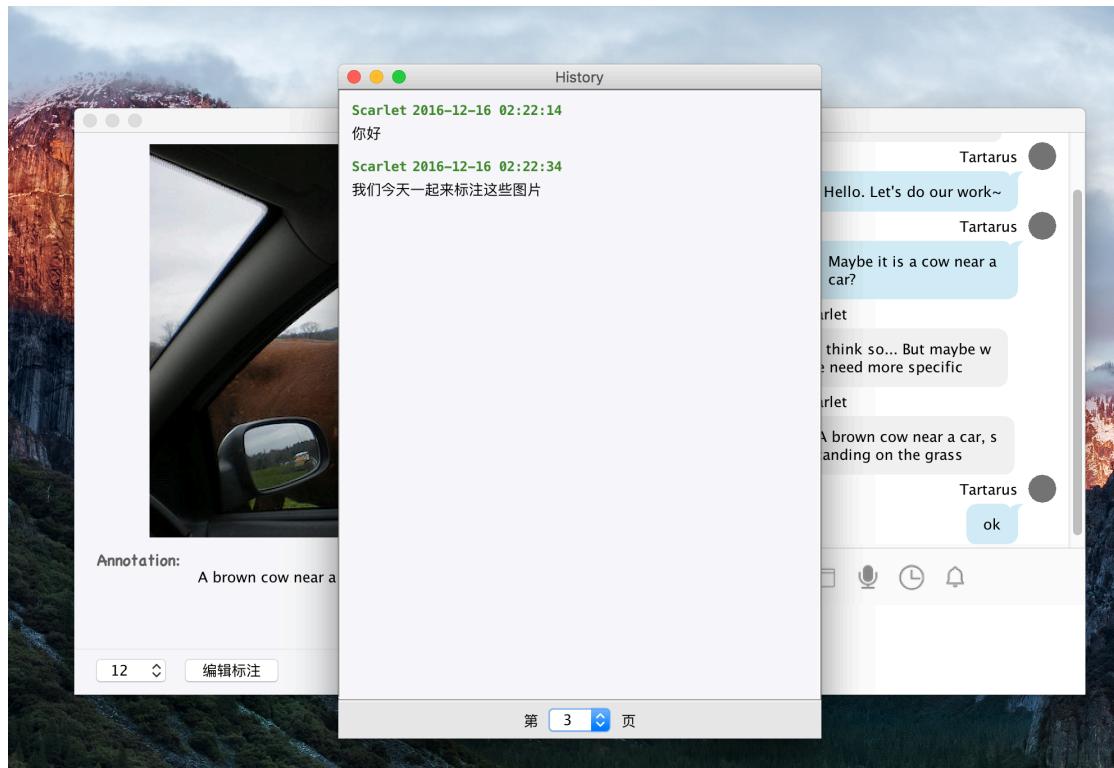


图 28 聊天记录界面 2

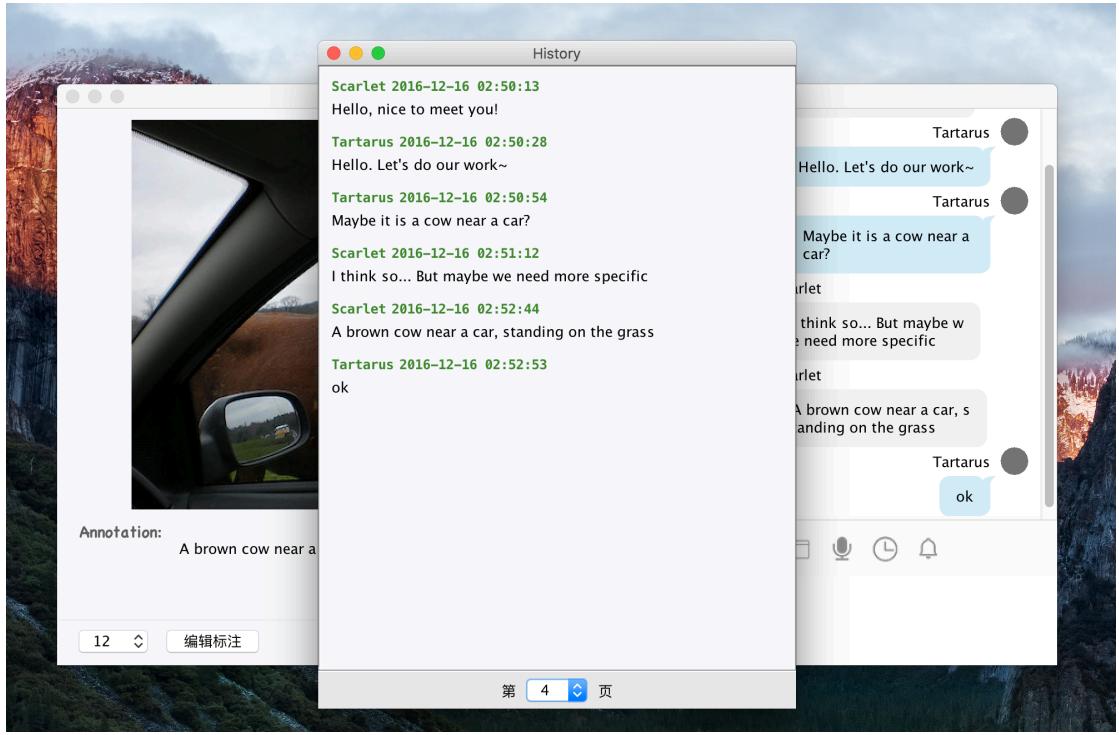


图 29 聊天记录界面 3

4.2.5 多人协作展示

在这里，我们以开两个客户端为例，首先，图片上时同步的，也就是两个标注者看到的一定是同一张图片：

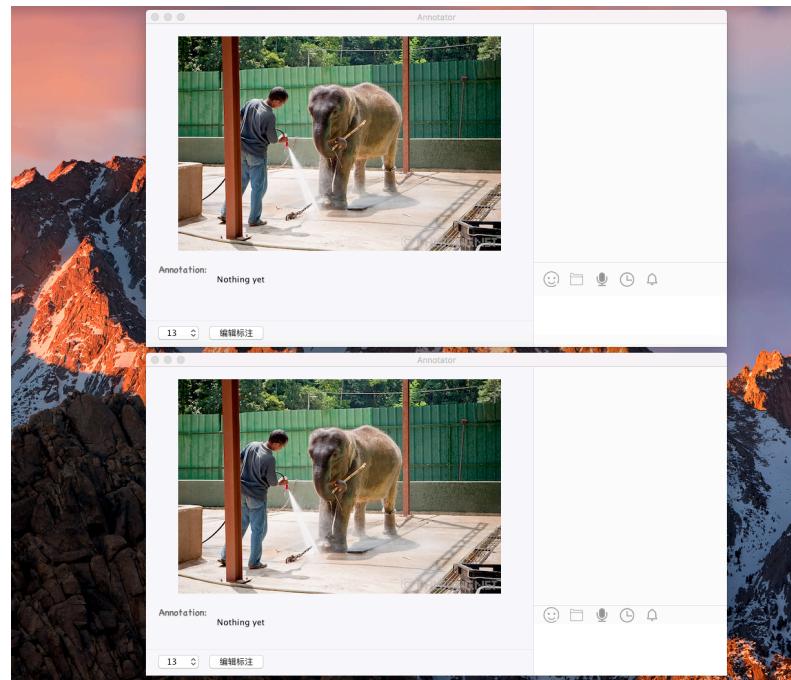


图 30 协同工作 1

两个标注者可以互相发信息讨论：

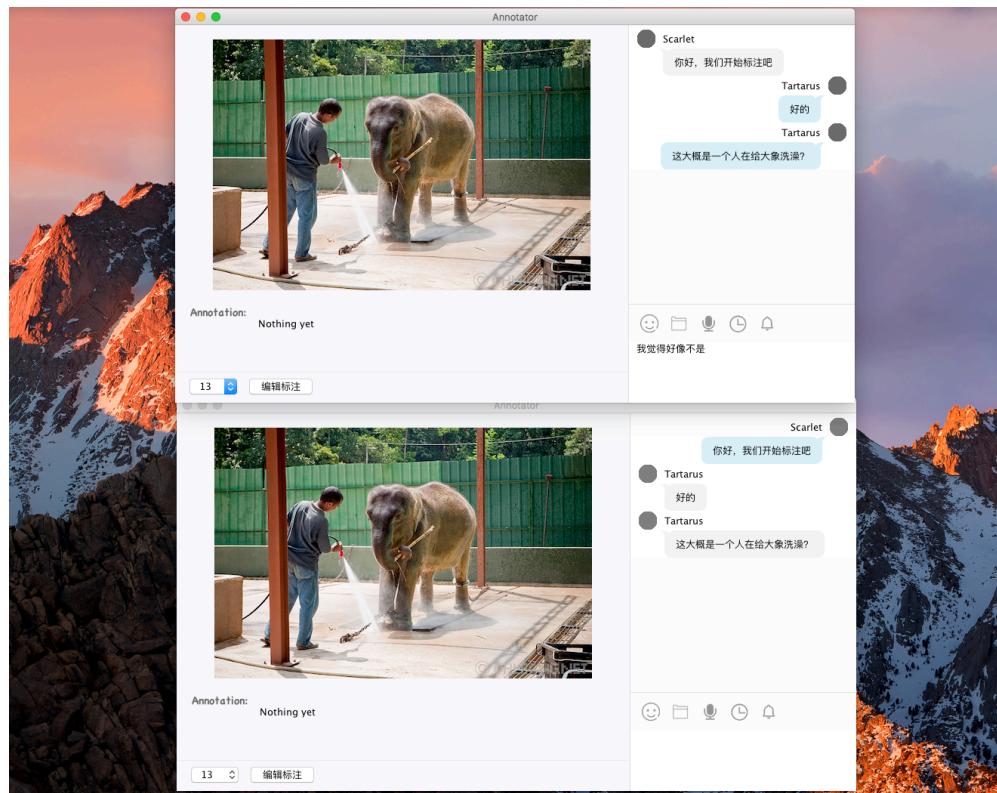


图 31 协作讨论

然后如果其中一个标注者进行了标注的修改，则这个标注结果会同步到另一个标注者的窗口中。

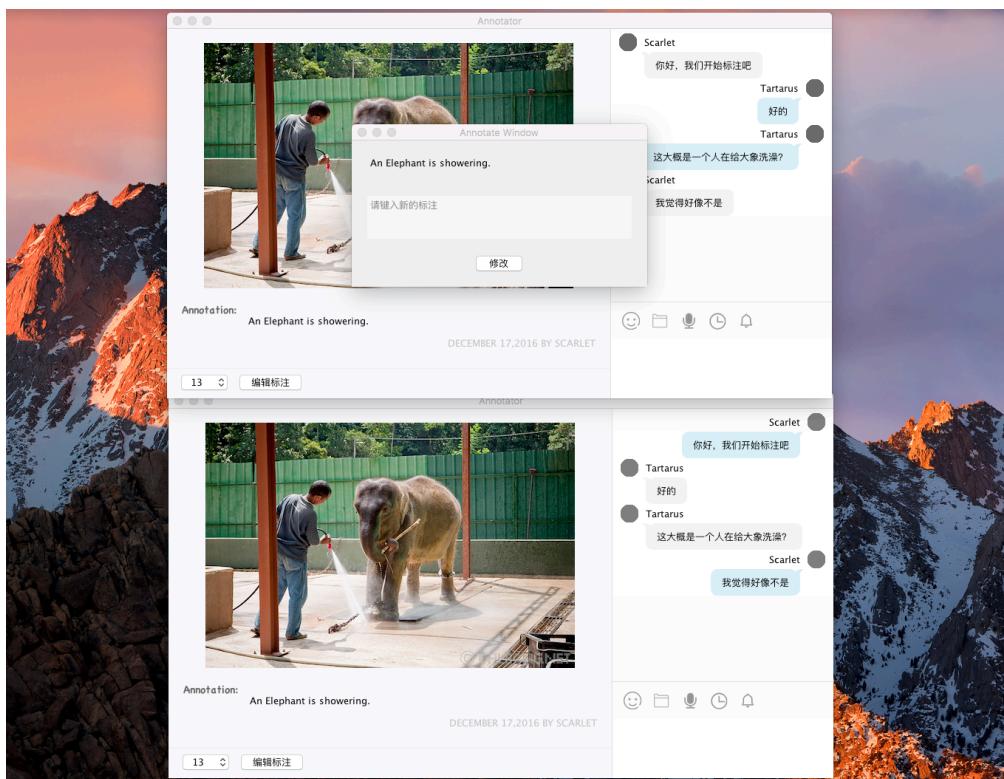


图 32 同步标注

5. 总结

总的来说，这次任务量要比以往几个程序和作业都要大，因为它同时涉及到 Socket 网络编程、基于 Swing 的 GUI 开发、SQLite 数据库编程、Java 的多线程并发。

首先在调试服务器和客户端之间的交互就比较困难，而由于自己总是粗心地没有将客户端和服务器的 DataIn 和 DataOut 相互对上，因此常常导致超时等待。但本次程序的编写过程中，我觉得把同步的请求和异步的请求分开大大地减小了我的工作量，因为之前试过同步请求和异步请求互相交织在一起导致各种各样的问题。

另外就 Java 的 Swing 进行 GUI 开发不是特别友好，因为本人曾经编写过安卓的 UI，因此很不习惯没有 xml 单独开来地进行布局和上色。另外就是 Swing 总是会出现写的代码跟显示的窗口跟自己所想的完全不一样，折腾了很久才明白怎么一回事。不过在编写 Swing 的过程中，确实学到了很多 GUI 的新知识，比如自己用 path 绘制自适应的聊天气泡等。

这次课程设计我做了我一直以来想做的一件事，因为自己曾在实验室做过相关工作，因此深知高质量标注的对于相关实验任务的处理非常有帮助。因为这次要做的课程设计不限制主题，所以完成着么一份工作也是非常开心的事情了。同时又掌握了 Java 服务端与客户端交互编程、GUI 的开发、JDBC 的使用、多线程的使用等。

参考资料

- [1] Oracle 官网.
- [2] SQLite JDBC 教程.
- [3] Google、stackoverflow.