实验报告

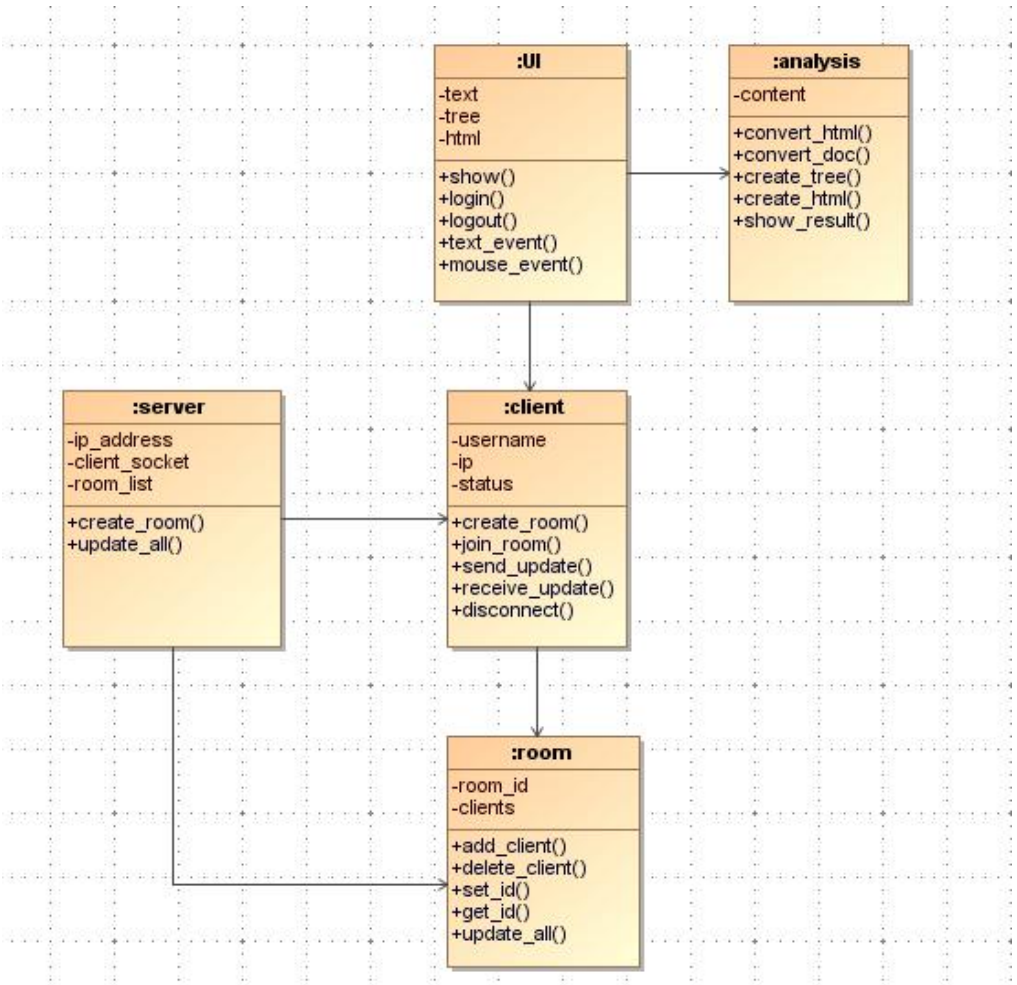
一、问题说明

实现一个 Markdown 编辑器，具有如下功能：

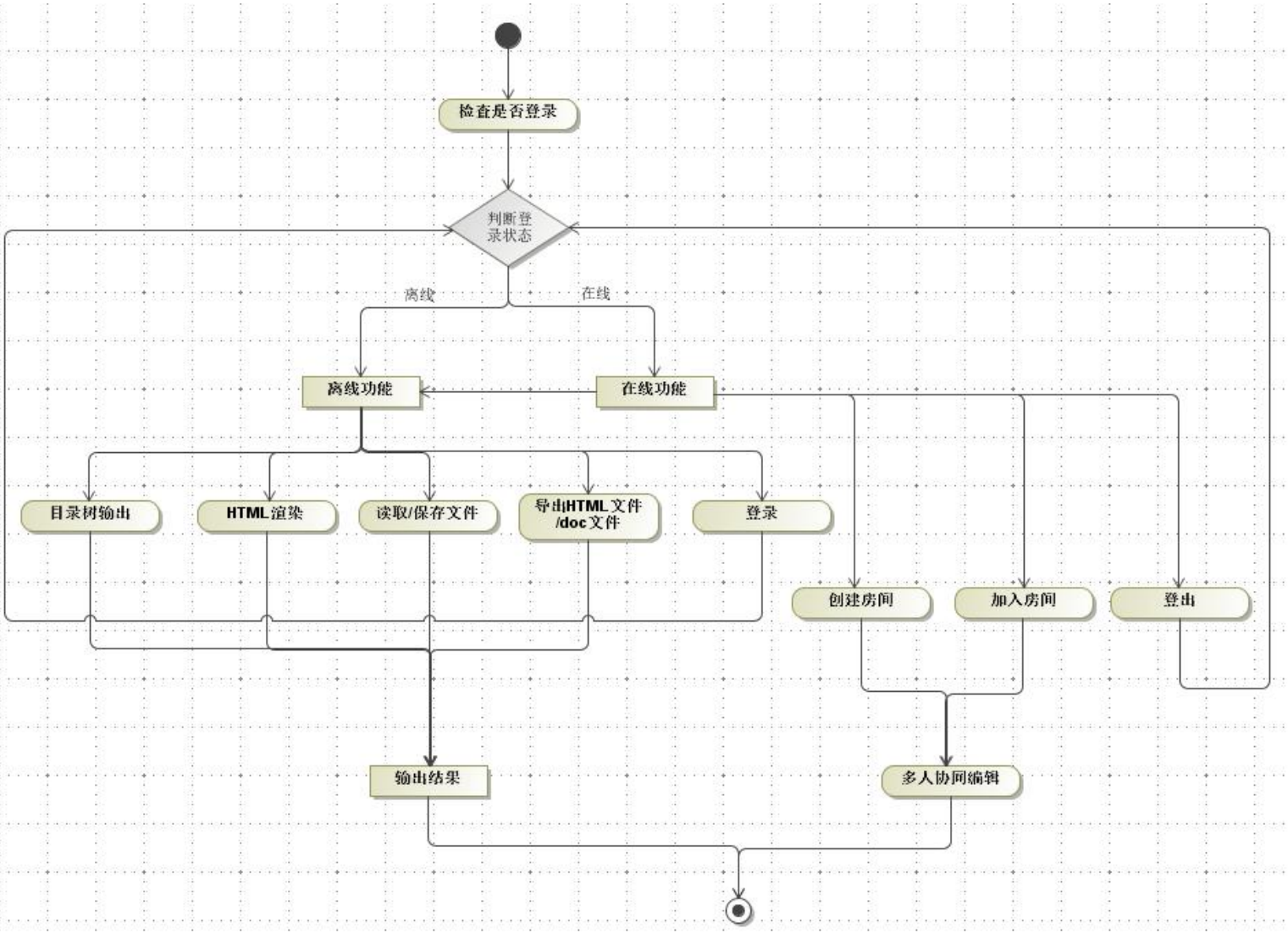
- 能编辑 Markdown 文档
- 能在编辑区的左侧看到实时的文档目录
- 能保存和打开 Markdown 文档
- 能输出 html
- 能建立一个网络服务，以供其他编辑器连接
- 能连接其他编辑器，连接后可编辑对方正在编辑的文档
- 连接了其他编辑器后，能实时同步反映服务器上的文件在其他编辑器上的修改
- 加分项（5 分）：能实时在编辑区右侧看 Markdown 渲染后的效果
-

二、实现方式

类图：



流程图：


检查是否登录
判断登录状态
离线
在线
离线功能
在线功能
目录树输出
HTML渲染
读取/保存文件
导出HTML文件/doc文件
登录
创建房间
加入房间
登出
输出结果
多人协同编辑

UI 类：

1. 定义用户界面，定义菜单栏，未登录状态下可以保存和读取文件，并能导出 HTML 和 doc 文件，同时提供在线登录功能，只有在登录状态下才提供房间创建和加入功能。

```
//文件菜单
{
    mFileMenu = new JMenu("文件");
    mFileMenu.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mMenuBar.add(mFileMenu);

    mOpenItem = new JMenuItem("打开");
    mOpenItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mFileMenu.add(mOpenItem);
    mOpenItem.addActionListener(this);

    mSaveItem = new JMenuItem("保存");
    mSaveItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mFileMenu.add(mSaveItem);
    mSaveItem.addActionListener(this);

    mExportHTMLItem = new JMenuItem("导出HTML");
    mExportHTMLItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mFileMenu.add(mExportHTMLItem);
    mExportHTMLItem.addActionListener(this);

    mExportDocItem = new JMenuItem("导出docx");
    mExportDocItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mFileMenu.add(mExportDocItem);
    mExportDocItem.addActionListener(this);


//登录和注册菜单
{
    mClientMenu = new JMenu("登录/注销");
    mClientMenu.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mMenuBar.add(mClientMenu);

    mLoginItem = new JMenuItem("登录");
    mLoginItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mClientMenu.add(mLoginItem);
    mLoginItem.addActionListener(this);

    mLogoutItem = new JMenuItem("注销");
    mLogoutItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mClientMenu.add(mLogoutItem);
    mLogoutItem.addActionListener(this);
}

{
    mRoomMenu = new JMenu("房间");
    mRoomMenu.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mMenuBar.add(mRoomMenu);

    mCreateRoomItem = new JMenuItem("创建房间");
    mCreateRoomItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mRoomMenu.add(mCreateRoomItem);
    mCreateRoomItem.addActionListener(this);

    mJoinRoomItem = new JMenuItem("加入房间");
    mJoinRoomItem.setFont(new Font("Microsoft YaHei", Font.PLAIN, 15));
    mRoomMenu.add(mJoinRoomItem);
    mJoinRoomItem.addActionListener(this);

    mRoomMenu.setVisible(false);
```

2. 添加鼠标事件，为不同的按钮定义动作：

```
//监听事件
@Override
public void actionPerformed(ActionEvent event) {
    Object item = event.getSource();

    //打开markdown文件
    if(item == mOpenItem) {
        String tmp = Utility.getContentFromExternalFile();
        if(tmp != null) {
            mText = tmp;
            mTextArea.setText(mText);
        }
    }

    //保存
    else if(item == mSaveItem) {
        if(mTextChanged) {
            if(Utility.saveContent(mText, "md"))
                mTextChanged = false;
        }
    }

    //导出HTML
    else if(item == mExportHTMLItem) {
        Utility.saveContent(Utility.getStyledHTML(mHTML, mCSS), "html");
    }
```

3. 目录结构，用了 JTree，生成关于目录的树状结构，并对每个节点添加点击事件：

```
    /**
     * 为目录更新标题
     */
    private void setTitles() {
        Pattern pattern = Pattern.compile("<h(\\d)>(.*?)</h(\\d)>", Pattern.CANON_EQ);
        Matcher matcher = pattern.matcher(mHTML);

        mRoot.removeAllChildren();
        while(matcher.find()) {
            int rank = matcher.group(1).charAt(0) - '0';
            String title = matcher.group(2);

            DefaultMutableTreeNode target = mRoot;
            for(int i = 1; i < rank; i++) {
                target = (DefaultMutableTreeNode)target.getChildAt(target.getChildCount() - 1);
            }
            mTreeModel.insertNodeInto(new DefaultMutableTreeNode(title), target, target.getChildCount());
        }
        mTree.updateUI();
    }
```

4. 编辑区，使用了 JTextArea，生成了编辑区模块：

```
    /**
     * 创建一个{@link JTextArea}域，用来编辑markdown文本
     */
    private void createTextArea() {
        mTextArea = new JTextArea();
        mTextArea.setLineWrap(true);
        Font font = new Font("Microsoft YaHei", Font.PLAIN, 18);
        mTextArea.setFont(font);
        mTextArea.getDocument().addDocumentListener(this);
        mTextArea.addMouseListener(this);
    }
```

5. HTML 渲染区，使用了 JEditorPane，用于显示渲染之后的内容：

```
/**
 * 创建一个{@link JEditorPane}域,用来显示HTML
 */
private void createEditorPane() {
    mEditorPane = new JEditorPane();
    mEditorPane.setContentType("text/html");
    mEditorPane.setEditable(false);

    HTMLEditorKit ed = new HTMLEditorKit();
    mEditorPane.setEditorKit(ed);

    mStyleSheet = ed.getStyleSheet();
    mStyleSheet.addRule("body {font-family:\"Microsoft YaHei\", Monaco}");
    mStyleSheet.addRule("p {font-size: 14px}");

    try {
        mHTML = mParser.parseMarkdownToHTML(mText);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Analysis 类：

1. Markdown 导出，主要用了开源的 markdown4j 包：

```
/**
 * 将content内容保存到外部文件
 * @param content
 * @return
 */
public static boolean saveContent(String content, String postFix) {
    JFileChooser jf = new JFileChooser();
    jf.setSelectedFile(new File("./untitled." + postFix));
    int option = jf.showSaveDialog(new JLabel());

    if(option == JFileChooser.CANCEL_OPTION)
        return false;

    File file = jf.getSelectedFile();

    try( PrintWriter writer = new PrintWriter(file) ) {
        writer.write(content);
    } catch(IOException e) {
        System.out.println("write error");
    }

    return true;
}
```

2. 文件读入，同样使用了开源包：

```
/**
 * 从外部文件读取文本内容
 * @return
 */
public static String getContentFromExternalFile() {
    JFileChooser jf = new JFileChooser();
    jf.showOpenDialog(new JLabel());
    File file = jf.getSelectedFile();
    if(file == null) return null;
    String text = null;

    try(BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                new FileInputStream(file), "utf-8"))) {
        StringBuilder sb = new StringBuilder();
        String tmp;
        while((tmp=in.readLine()) != null) {
            sb.append(tmp);
            sb.append("\n");
        }

        text = sb.toString();
    } catch (IOException e) {
        System.out.println("read error");
    };
    return text;
}
```

3. 实时显示，为了减少不必要的计算，这里用线程阻塞的方式，实现高效地更新。每次，想要更新的线程先 sleep 1000 毫秒，如果之后又有想要更新的线程，就将上一个阻塞，这样的话上一个就直接返回，放弃更新的操作。这样的效果就是，只要持续输入或删除内容，对文本内容做出改变的间隔不超过 1 秒钟，就一直不会更新，当用户停止 1 秒后，才会执行更新操作。大大减少了无用的计算：

```
private Thread lastThread = null;       //存储上一个要更新UI的线程，供下一个阻塞
private void updateUIEfficiently() {
    new Thread(() -> {
        Thread last = lastThread;
        lastThread = Thread.currentThread();

        try {
            //阻塞上一个更新的线程
            if(last != null) {
                last.interrupt();
            }
            Thread.sleep(1000);
        } catch(InterruptedException exc) {
            return;
        }

        if(Thread.currentThread().isInterrupted()) return;
        SwingUtilities.invokeLater(() -> {update();});

        if(isOnline&&(!isOther)) {
            String updation = mTextArea.getText();
            try {
                mClient.disposeRequest(RequestType.UPLOAD_UPDATION, updation);
            } catch (Exception e) {
                e.printStackTrace();
                Utility.error("与服务器端连接出现错误！");
            }
        }
        isOther=false;
    }).start();
```

Socket 编程部分：

1. 首先创建一个本地服务器 Server，开放 2 个端口，8080 和 8081。同时可以创建若干个 Client。Server 处理客户端发来的所有请求，并返回回复；Client 用于在客户端帮助用户发送所有请求。工程建立在 **C/S** 架构上。

```
/**
 * 建立服务器
 */
private void setServer() {
    new Thread(() -> {
        try {
            mServerSocket = new ServerSocket(SERVER_PORT);

        } catch (IOException e) {
            e.printStackTrace();
            Utility.error("本地服务器创建失败！");
        }

        //服务器不断接收客户端发来的连接请求，然后建立连接
        try {
            while(true) {
                //有新的客户端发来连接请求
                Socket socket = mServerSocket.accept();

                Client client = new Client();
                client.setSocket(socket);

                //新开一个线程
                new DisposeThread(client, 1).start();
            }
        } catch(Exception e) {
            e.printStackTrace();
            Utility.error("连接超时，或通路出现错误！");
        }
    }).start();
```

2. 所有客户端一旦试图进行登录或注册，就会创建一个 Client，并建立相应的端口进行通信。服务器也会新开一个线程对这个客户端进行服务。客户端进行的一切操作都是在向服务器发送请求，然后得到服务器的回复。当客户端退出登录或是退出程序，就断开连接的端口，释放资源，防止内存浪费。

```
/**
 * 连接服务器
 * @throws UnknownHostException
 * @throws IOException
 */
public void connectServer() throws UnknownHostException, IOException {

    socket = new Socket();
    socket.connect(new InetSocketAddress(SERVER_IP, SERVER_PORT),
                   CONNECT_TIME_OUT);

    mStreamReader = socket.getInputStream();
    mStreamWriter = socket.getOutputStream();

    mPrintWriter =
        new PrintWriter(
            new BufferedWriter(
                new OutputStreamWriter(
                    mStreamWriter)), true);

    mBufferedReader =
        new BufferedReader(
            new InputStreamReader(
                    mStreamReader));
}
```

3. 一旦一个用户创建或是加入了一个房间，那么服务器就必须通过另一个端口——8081，来进行房间内内容的更新等操作。所以加入房间后，客户端会试图连接服务器的 8081 端口。对应的服务器上 8081 端口专门有一个线程来进行处理，依然是每有一个客户端连接过来就新开一个线程进行服务。这个端口做的事就是专门把房间主人发过来的文本内容发送给房间其他所有成员让他们进行同步。所以，每个客户端也必须开一个线程，不断尝试从对应 8081 端口的 socket 读取内容，然后更新。

```
private void disposeCreateRoom(RequestType type) throws Exception {
    String request = type + "#";
    mPrintWriter.println(request);

    //接收回复
    String respond = mBufferedReader.readLine();
    String[] s = respond.split("#");

    if(!s[0].equals(SignInfo.SUCCESS.toString()))
        throw new Exception(respond);

    int id = Integer.parseInt(s[1]);

    try {
        connectServer2();

        request = RequestType.CONNET_ROOM_SERVER + "#" + name;
        mPrintWriter2.println(request);

        respond = mBufferedReader2.readLine();
        s = respond.split("#");

        if(s[0].equals(SignInfo.SUCCESS.toString())) {
            this.roomID = id;
        }
        else throw new Exception();
    } catch(Exception e) {
        e.printStackTrace();
        throw new Exception("连接房间服务器失败！");
    }
}
```

4. 关于请求的类型定义在 RequestType.java 中：

```
1  package util;
2
3  public enum RequestType {
4      LOGIN,
5      CONNET_ROOM_SERVER,
6      CREATE_ROOM,
7      JOIN_ROOM,
8      ASK_FOR_CHANGING,
9      UPLOAD_UPDATION,
0      CUT_CONNECT
1  }
```

Room 类：

1. 保存房间 ID，房主的 IP 信息，房间成员的信息：

```
public class Room {
    private int mID;

    public Room(int id) {
        mID = id;
    }

    private Client mHost;   //房间创建者
    private ArrayList<Client> mClients = new ArrayList<>();      //房间内的所有用户

    public Client getHost() {
        return mHost;
    }
    synchronized public void setHost(Client Host) {
        this.mHost = Host;
        add(mHost);
    }

    public int getID() {
        return mID;
    }

    synchronized public void add(Client client) {
        mClients.add(client);
    }
```

2. 创建房间，房间序号自动递增，最多为 100：

```
        /**
         * 处理创建房间请求
         * @throws Exception
         */
        private void disposeCreateRoom() throws Exception {
            if(mRooms.size() == MAX_ROOM_NUMBER) {
                mPrintWriter.println("很抱歉！服务器较拥挤，不能创建房间！");
            }
            else {
                int id = 0;
                //找到第一个空的位置，分配一个房间id
                for( ; id < MAX_ROOM_NUMBER; id++) {
                    if(mRooms.get(id) == null)
                        break;
                }

                Room room = new Room(id);
                room.setHost(mClient);
                mClient.setRoomID(id);
                mRooms.put(id, room);

                mPrintWriter.println(SignInfo.SUCCESS + "#" + id);
            }
        }
```

3. 加入房间：

```
/**
 * 处理加入房间请求
 * @param id 要加入的房间的id
 * @throws Exception
 */
private void disposeJoinRoom(String idString) throws Exception {
    int id;
    try {
        id = Integer.parseInt(idString);
    } catch(NumberFormatException e) {
        mPrintWriter.println("房间id必须是0~99的数字！");
        return;
    }
    Room room = mRooms.get(id);
    if(room == null) {
        mPrintWriter.println("您要加入的房间不存在！");
    }
    else {
        room.add(mClient);
        mClient.setRoomID(id);

        mPrintWriter.println(SignInfo.SUCCESS + "#");
    }
}
```

4. 房间内共享更新内容，每一次有一个用户进行修改之后都将修改后的内容上传至服务器，再通过服务器发送给房间内的另外所有用户，实现编辑内容的实时共享：

```
    /**
     * 把更新发送给组内所有其他成员
     * @param updation
     * @throws Exception
     */
    synchronized public void updateAllMember(String updation) throws Exception {
        int size = mClients.size();
        for(int i = 0; i < size; i++) {
            Client client = mClients.get(i);
            client.getPrintWriter2().println(updation + "\n" + "$$END$$");
        }
    }
}
```

所有房间内的用户在收到服务器的更新指令后会自动刷新 UI 显示的内容，但是必须使用一个布尔值变量，用于表示这一次的更新是由他人发起的还是由自身发起的，如果由他人发起的修改就不再上传修改内容至服务器，如果是自身修改就上传至服务器，否则会引起循环更新的问题。

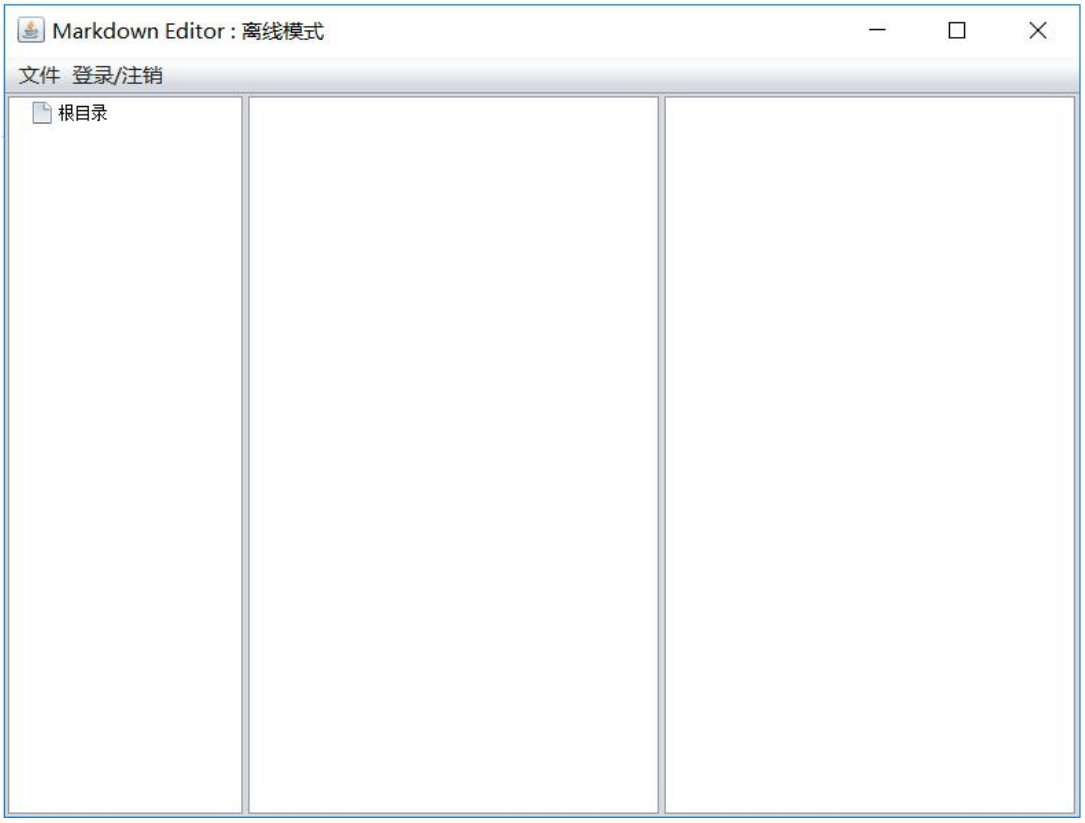
```
        /**
         * 当加入一个房间后，就要开始一直监测服务器发来的信息
         */
        private void startUpdateMonitor() {
            new Thread(() -> {
                try {
                    while(true) {
                        String updation = mClient.startMonitor_getUpdation();
                        isOther=true;
                        SwingUtilities.invokeLater(() -> {
                            mTextArea.setText(updation);
                        });
                    }
                } catch(Exception e) {
                    e.printStackTrace();
                    Utility.error("与服务器连接中断！");
                    return;
                }
            }).start();
        }


    if(isOnline&&(!isOther)) {
        String updation = mTextArea.getText();
        try {
            mClient.disposeRequest(RequestType.UPLOAD_UPDATION, updation);
        } catch (Exception e) {
            e.printStackTrace();
            Utility.error("与服务器端连接出现错误！");
        }
    }
    isOther=false;
}).start();
}
```

三、实验截图

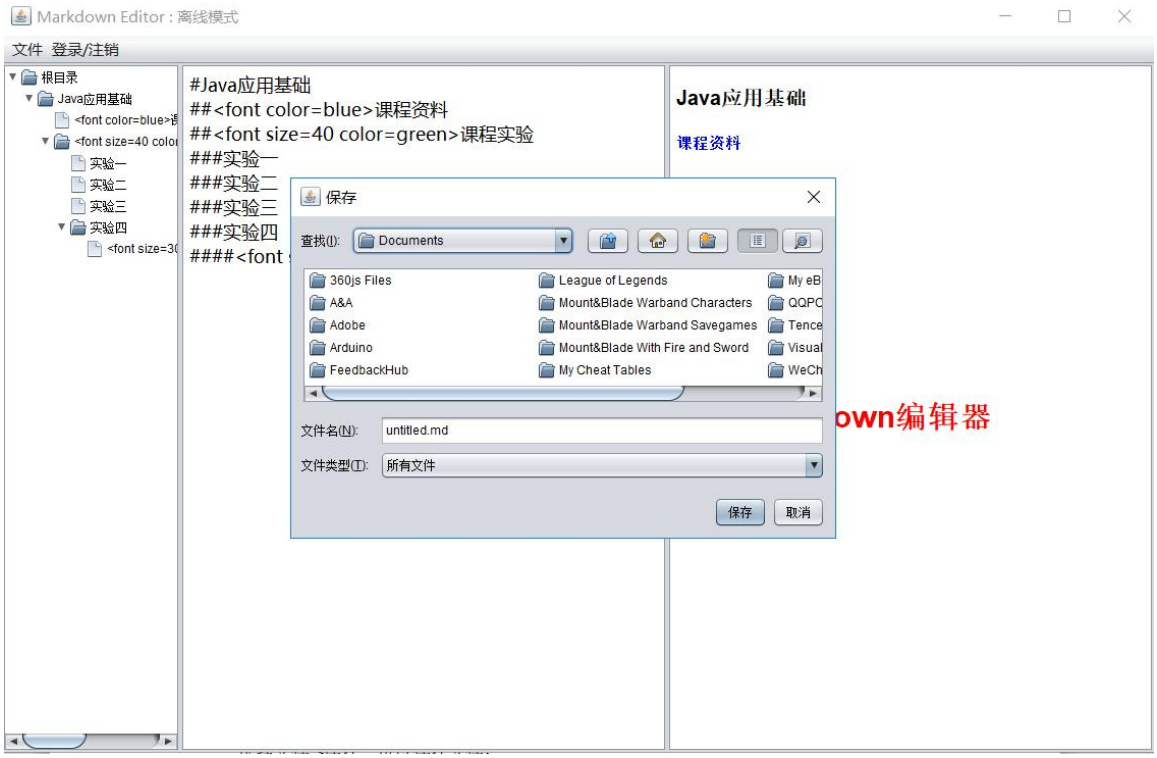
离线状态：

主页面：

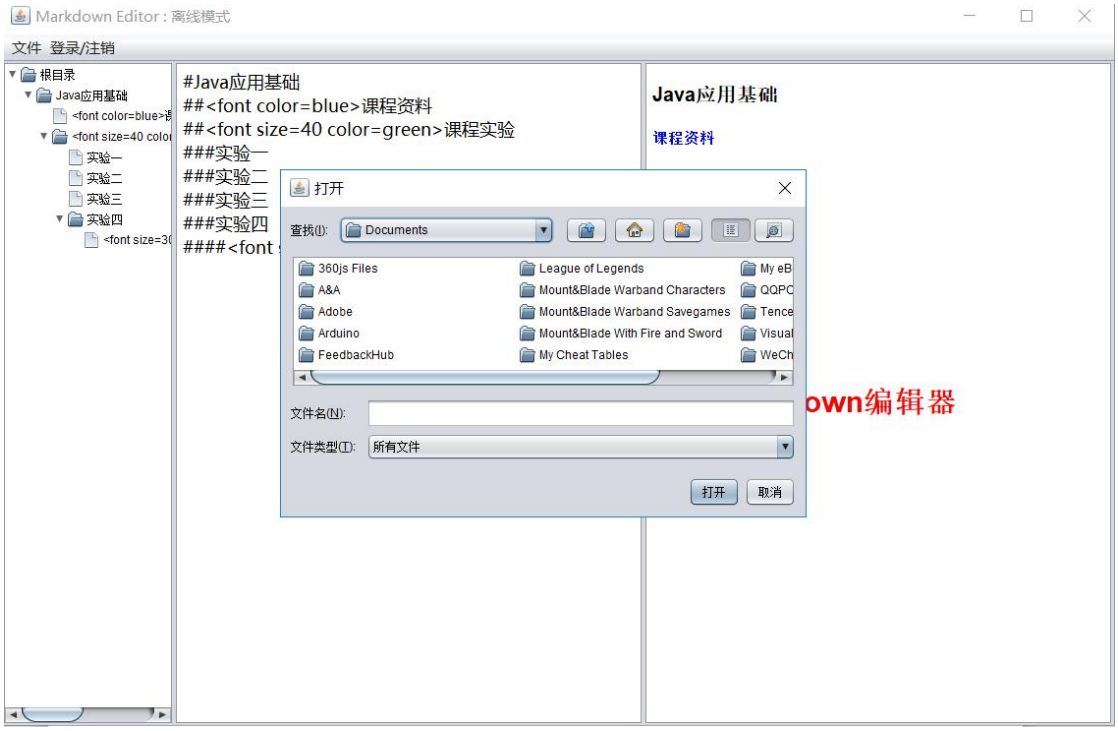


添加一些内容，显示目录树结果与修饰结果：

选择文件->保存，可以保存文件：



选择文件->打开，可以读取文件：



选择文件->导出为 HTML 可以保存为 HTML 文件：

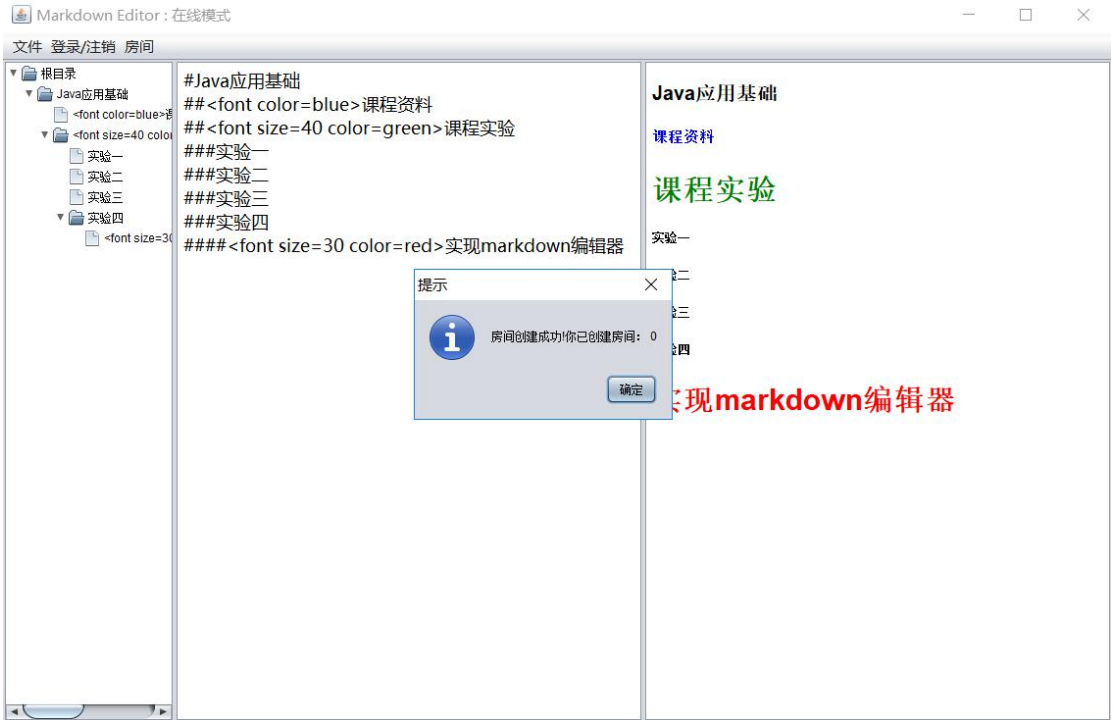
在线状态：

首先需要启动服务器：

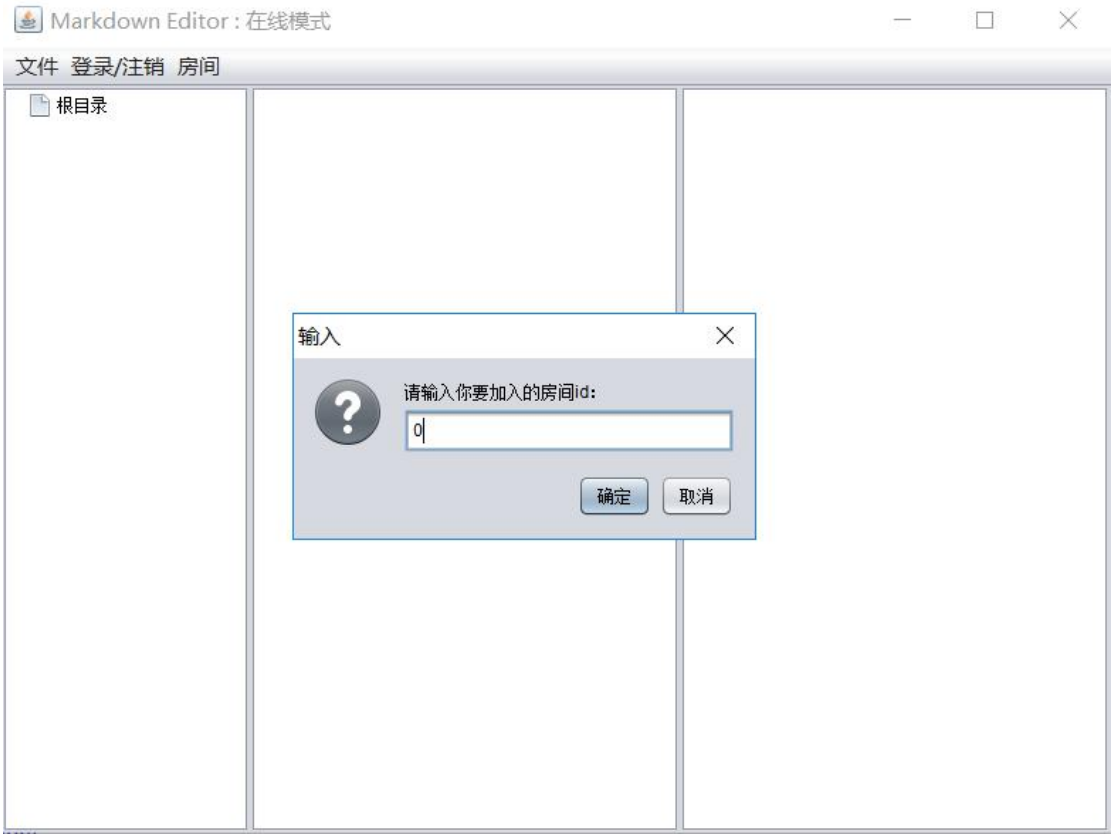
```
51    public static void main(String[] args)
52    {
53        MainServer server = new MainServer();   //开始主服务器运行
54        System.out.println("Main server is starting!");
55    }
```

在客户端上点击登录按钮切换为在线模式：

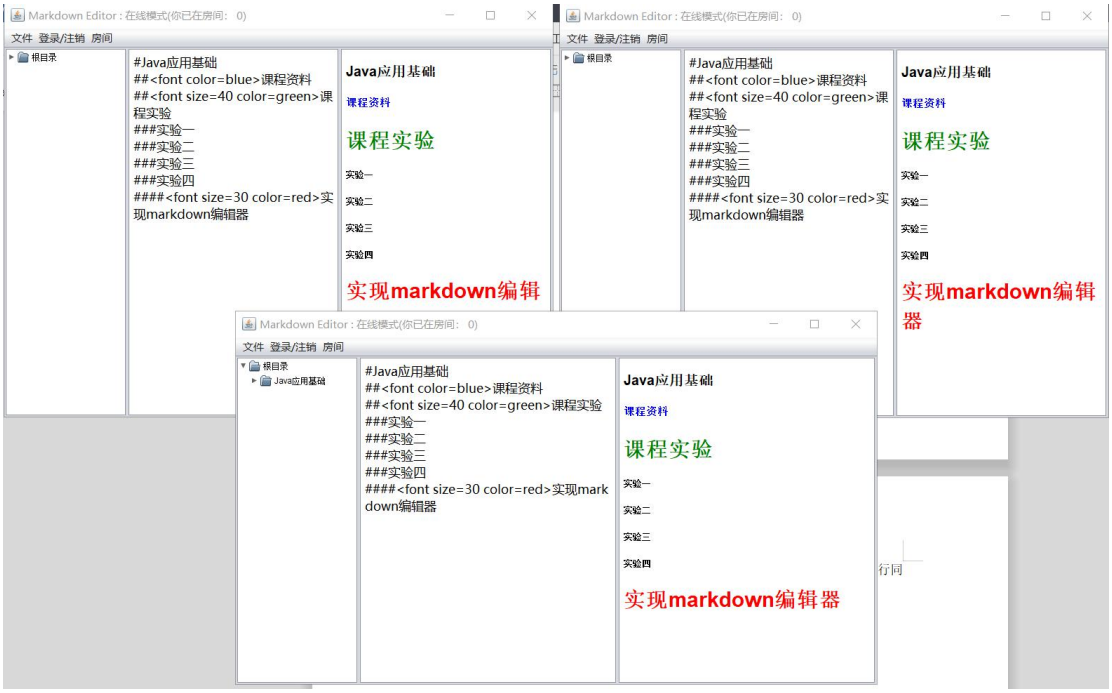
点击房间->创建房间，可以创建一个新房间：

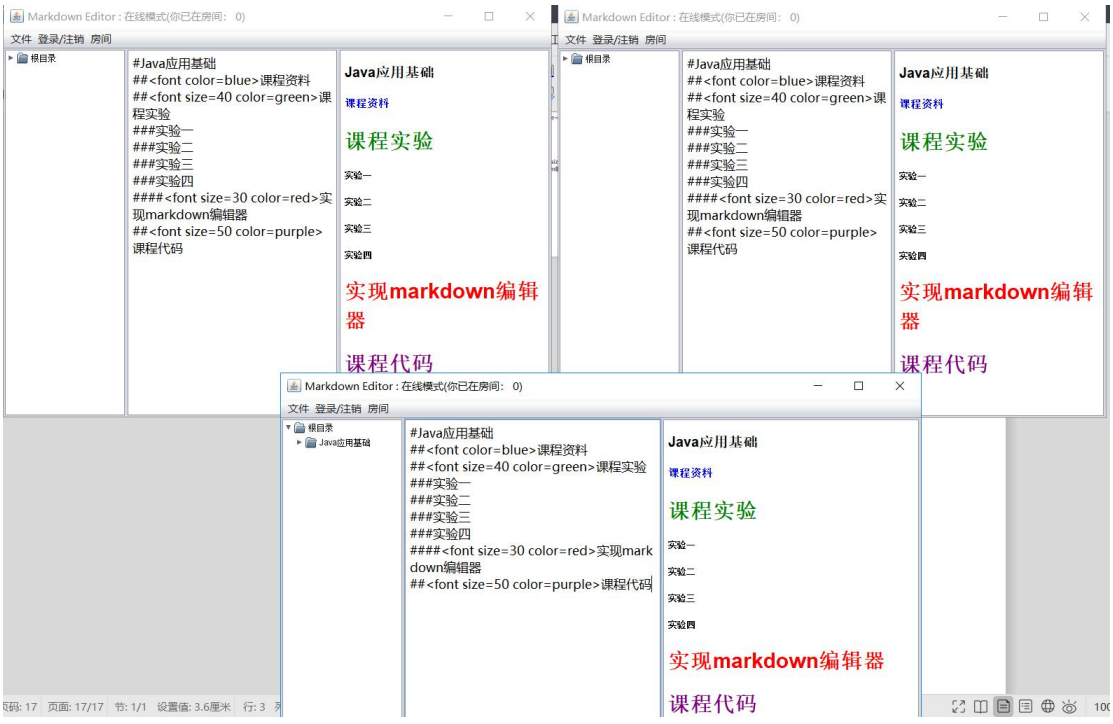


创建大量客户端，登录之后选择房间->加入房间，输入正确房间号之后可以加入房间：



每一个客户端都可以共同对内容进行修改，修改的内容会在其他客户端上进行同步：





在登录状态下点击注销，会退出在线模式，返回离线模式：