聊天室: 运用设计模式进行重构

项目简介

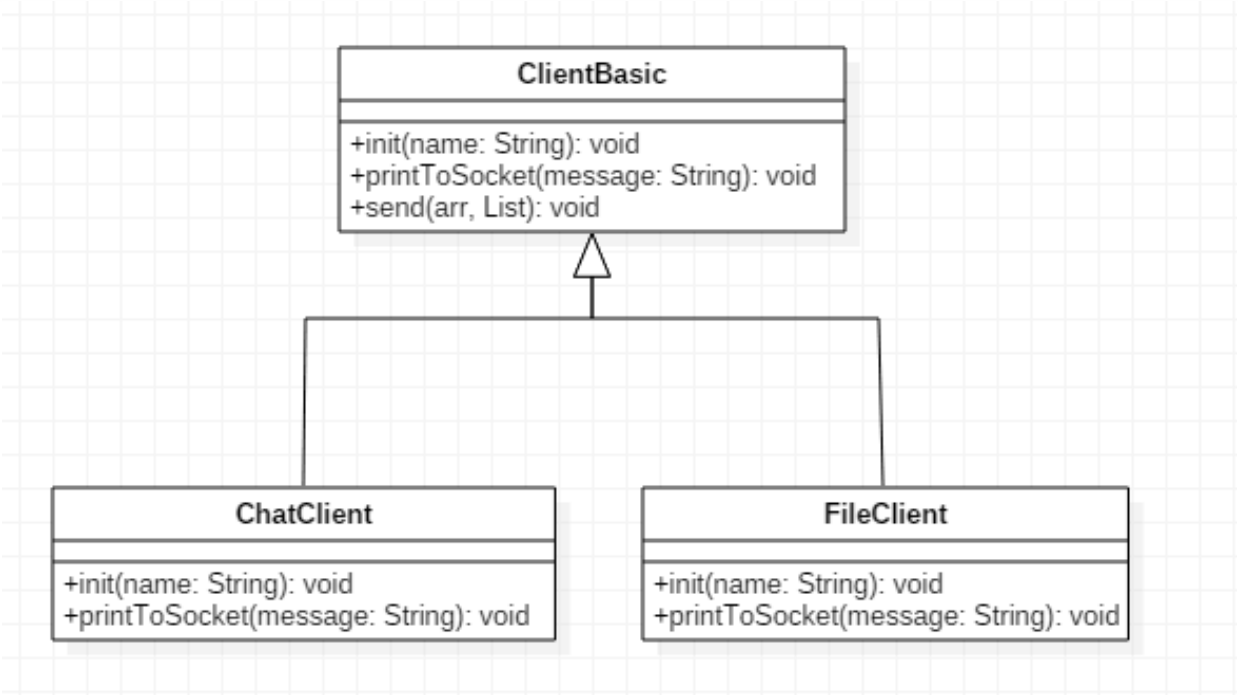
本项目实现的是一个在线聊天室，实现在线用户之间信息交换的功能。

项目重构概况

我们项目的主要目标是实现抽象和解耦，具体用到了 工厂模式，模版模式，单例模式，过滤器模式，策略模式，装饰器模式；
改进了代码结构和代码风格，去除代码中的硬编码部分，修正原有程序中的bug；
同时利用sonarlint检验代码质量，控制代码质量。
下面依次对上述改进做简要描述。

模版模式

因为ChatClient和FileClient里面的方法太过于相似了，都是构造、发送、监听、运行、停止等，所以抽象出了一个ClientBasic基类，其中的send方法包含init初始化和print发送到socket两个部分，所以定义为模板方法，两个子类只需要重载init和printToSocket两个函数即可。

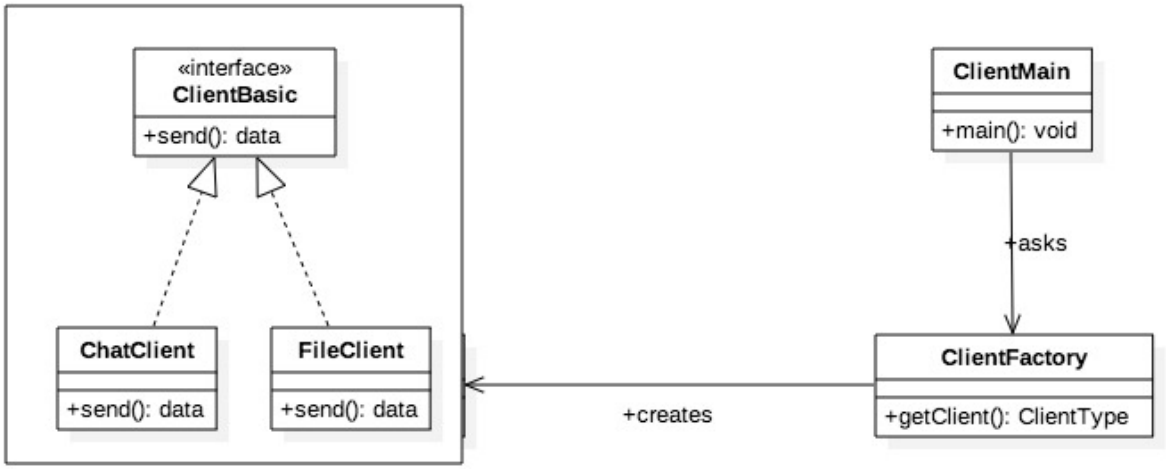



```
//ClientBasic.java：基类代码
abstract void init(String name);
abstract void printToSocket(String message);
public final void send(String...arr){
    if(arr.length>1)
        init(arr[1]);
    printToSocket(arr[0]);
  }


//FileClient.java：FileClient中的重载
@Override
  void init(String name) {
    // TODO Auto-generated method stub
    try{
        file = new File(name);
        fis = new FileInputStream(file);
    }catch(Exception e){
        System.out.println("init error!");
    }

  }

  @Override
  void printToSocket(String message) {
    // TODO Auto-generated method stub
    try{
        sender.writeUTF(message);
        sender.writeLong(file.length());
        byte[] buff = new byte[1024];
        int length = 0;
```

```
            while ((length = fis.read(buff, 0, buff.length)) > 0) {
              sender.write(buff, 0, length);
              sender.flush();
            }
        }catch(Exception e){
            System.out.println("print to socket error!");
        }
      }

    //ChatClient.java：ChatClient中的重载
    @Override
      void init(String name) {
        // TODO Auto-generated method stub

      }

      @Override
      void printToSocket(String message) {
        // TODO Auto-generated method stub
        sender.println(message);
        sender.flush();
      }

```

简单工厂模式

客户端交互分发送文件和发送信息两种类型，对应着ChatClient和FileClient两个类，他们都继承自同一个名字为ClientBasic的抽象类，那么我们就可以创造一个工厂类，对客户端屏蔽创建逻辑，只是通过一个共同的接口来新建对象，如factory.getClient(ClientName);,如果我们将来有发送图片或者发送视频，也可以只修改工厂创建接口就可以方便的调用。



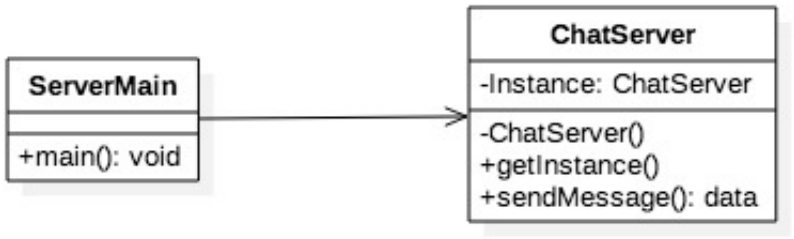
```java
    //ClientFactory.java：工厂代码实现
    public ClientBasic getClient(String strType,String serverIP, int port, String username, Client parentThread) throws IOException{
        if(strType == null)return null;
        else if(strType.equals("ChatClient")) return new ChatClient(serverIP,port,username,parentThread);
        else if(strType.equals("FileClent")) return new FileClient(serverIP,port,username,parentThread);
        return null;
    }

    //调用
    chatClient = clientFactory.getClient("ChatClient", ip, port, username,this);
    fileClient = clientFactory.getClient("FileClient", ip, port, username, this);
```

问题1:
在从工厂中获取对象的时候如果单词拼错就无法获取到对象，我们在重构过程中就是一不小心少写了一个i而导致系统无法正常运转，并且一直无法找到Bug的来源。
改进1
取消字符串的输入，改用数字012...代替，出错率大大降低。
问题2
虽然采用数字可以有效降低出错率，但是可读性很差，获取对象时并不能确保得到的就是要的client类型
改进2
提供多个工厂方法，分别用来创建不同的对象

```java
    //  获取chatclient
    public ClientBasic getChatClient(String serverIP, int port, String username, Client parentThread) throws IOException{
        return new ChatClient(serverIP,port,username,parentThread);
    }

    //  获取fileclient
    public ClientBasic getFileClient(String serverIP, int port, String username, Client parentThread) throws IOException{
        return new FileClient(serverIP, port, username, parentThread);
    }


    //调用
    chatClient = clientFactory.getChatClient(ip, port, username, this);
    fileClient = clientFactory.getFileClient(ip, port + 1, username, this);
```

单例模式

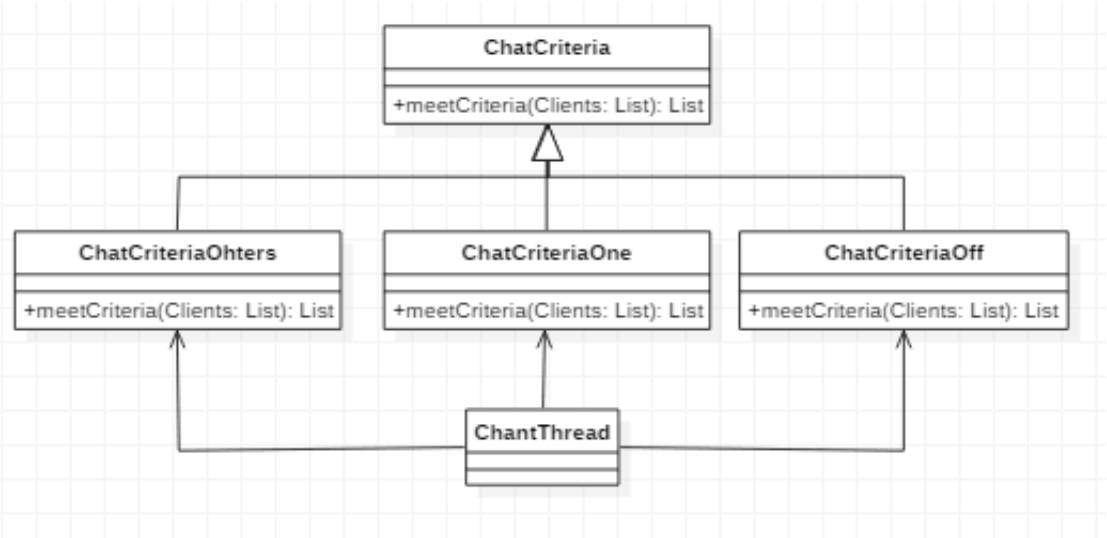
主服务器负责处理数据的有FileServer和ChatServer两个类，分别处理文件传输和信息传输，监听端口分别为8080和8081，一台服务器上只能存在一个FileServer和ChatServer，否则就会导致系统紊乱，在每次创建子线程的时候，子线程中都要有一个FileServer或者ChatServer，为了确保安全，决定采用单例模式。




```
    //ChatServer.java：ChatServer单例模式实现
    public static ChatServer getChatServer() throws IOException {
        if(instance == null)
            instance = new ChatServer(port);
        return instance;
    }

    //FileServer.java：FileServer单例模式实现
    public static FileServer getFileServer() throws IOException {
        if(instance == null)
            return new FileServer(port);
        return instance;
    }
```

过滤器模式

在服务器上运行的程序有一个数组就是客户端集，针对这同一个数组集合在不同的条件下往往有不同的标准，如在群聊中需要过滤掉自己，在P2P聊天中需要过滤掉所有的其他人，在上线时需要获得所有的在线用户信息，在下线时需要过滤掉所有不包含自己的客户端。那么我们就可以使用过滤器模式，构造一个接口，其中有一个过滤的函数，在不同的情况下可以有不同的实现在获取自己想要的子集。



```
    //ChatCriteria.java：ChatCriteria接口代码实现
    public interface ChatCriteria {
        public List<ChatThread> meetCriteria(List<ChatThread>
clients,String username);
    }

    //ChatCriteriaOne.java：ChatCriteraOne代码实现
    public class ChatCriteriaOne implements ChatCriteria{
    @Override
    public List<ChatThread> meetCriteria(List<ChatThread> clients,
String username) {
        List<ChatThread> meet = new ArrayList<ChatThread>();
        for(ChatThread chatThread : clients){
            if(chatThread.getUsername().equals(username)){
                meet.add(chatThread);
                break;
            }
        }
        return meet;
    }

    //ChatCritiaOther文件：代码实现
    //...

    //调用
    List<ChatThread> others = chatCriteriaOthers.meetCriteria(clients,
this.getIdentifier());

```

策略模式

在我们的模拟聊天室中，消息类型分为用户上线，用户下线群发消息，私聊消息，发送文件五种，通过策略模式将一个系列的算法包装到一系列的策略类里面，从而使得对应于每个消息类型的算法可以在不影响到客户端的情况下发生变化。 下面简单展示策略模式的实现

```java
    //  ClientView.java: 策略模式-消息类型接口
    public interface updateGUI {
        public void updateGUI(String command, String message, String
sender);
    }

    //  策略模式-群聊
    public class updateForGroup implements updateGUI {
        @Override
        public void updateGUI(String command, String message, String
sender) {
            if (chatUser.equals("GroupChat")) {
                receiveMessage(sender, message);
            } else {
                String name = (String)listModel.elementAt(0);
                listModel.remove(0);
                listModel.add(0, name + "(New Message)");
            }
            return;
        }
    }

    //  策略模式-私聊...
```

装饰器模式

我们的BufferedReader和PrintWriter就是装饰模式，在数据传输时，通过BufferedReader来不断监听socket端口，当我们需要发送数据时，就通过PrintWriter网Socket里面传输数据。对应的文件传输的DataOutputStream和DataInputStream也是装饰模式。 由于Java的IO需要很多性能的各种组合，如果这些性能都是用继承的方法实现的，那么每种组合都需要一个类，这样就会造成大量性能重复的类出现，装饰模式可以尽可能地解决这些问题。在使用IO时，Java的IO是由一些基本的原始流处理器和围绕它们的装饰流处理器所组成的。

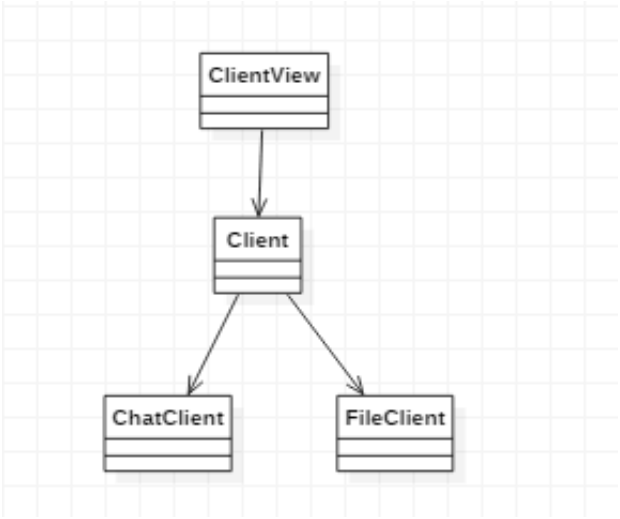
硬编码部分改进

由于项目成员之间使用mac和windows的不兼容性，使得项目代码在重构的过程中由于文件格式的不同时常会发生乱码，同时，在GUI层过多的菜单内容和消息提示内容内嵌在代码中，要更改的十分困难，所以我们通过声明常量来解决硬编码问题。

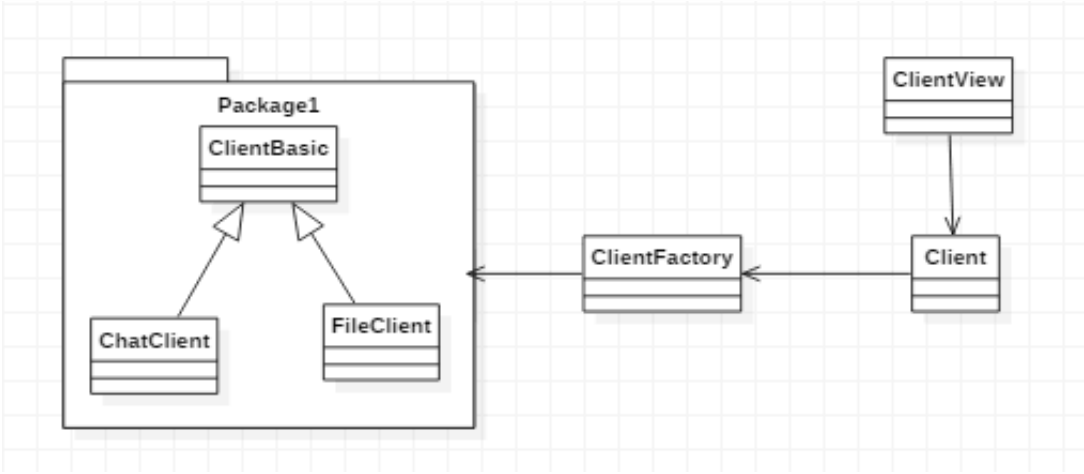
```
    //  定义各项常量
    public class constant {
        public static final String emptyIpPort = "IP地址和端口不能为空";
        public static final String emptyUser = "用户名不能为空";
        public static final String emptyMessage = "消息不能为空";
        public static final String illegalUsername = "用户名不能有[#]字
符";
        public static final String port = "8080";
        public static final String connect = "连接";
        public static final String exit = "退出";
        public static final String send = "发送";
        public static final String sendFile = "发送文件";
        public static final String ip = "端口";
        public static final String hostIP = "服务器IP";
        public static final String userName = "姓名";
        public static final String connectInfo = "连接信息";
        public static final String messageContext = "聊天消息";
        public static final String onlineUser = "在线用户";
        public static final String title =  "有聊";
        public static final String picUrl = "chat.png";
        public static final String systemMessage = "系统消息";
    }
```

代码结构改进

重构前客户端代码结构

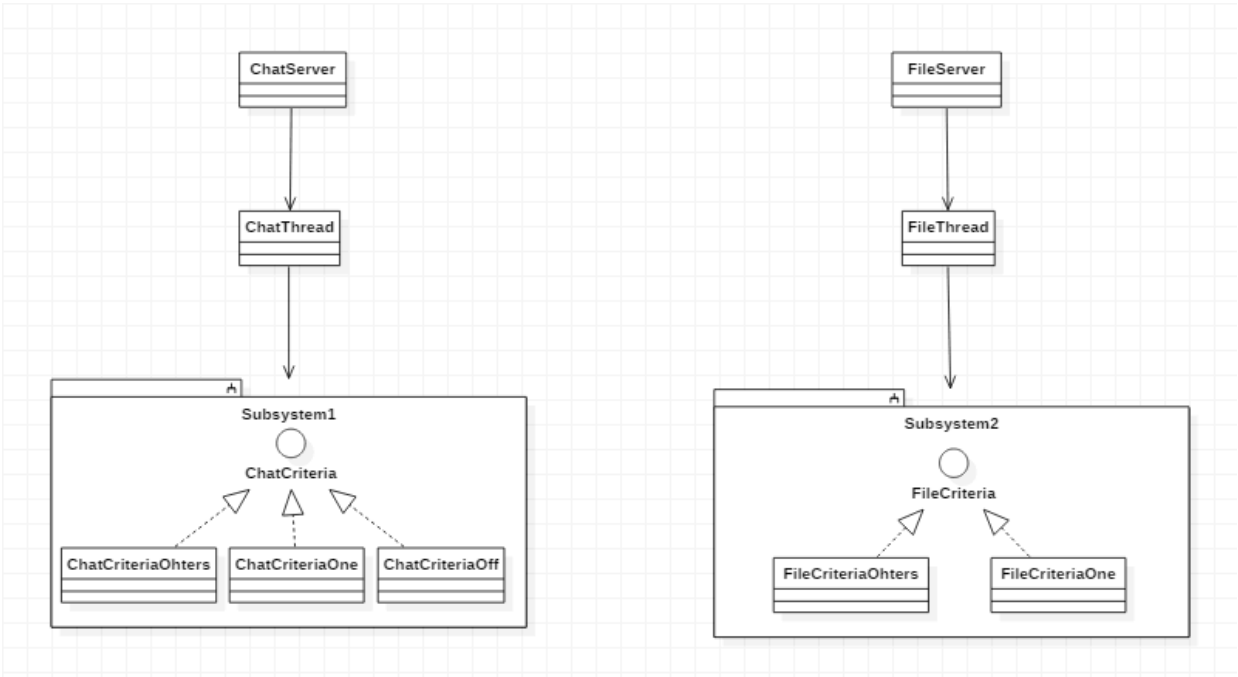


重构后客户端代码结构



重构前服务器端代码结构

重构后服务器端代码结构



修正bug

1. 多次点击连接会多次添加用户
2. 收到多条消息会有多个New Message提示
3. 点击关闭按钮实际上没有退出

sonarlint代码质量监控

由于最开始完成项目的紧迫性，只是完成了对应的功能而没有着重于代码质量的管理，通过在eclipse中安装sonarlint插件，从不遵循代码标准，潜在的缺陷，重复，注释不足或者过多等多方面进行项目代码质量的管理。 我们在项目进行过程中通过sonarlint对项目潜在的问题和缺陷在第一时间处理掉。

部署启动

进入到项目的主目录之后，按照以下步骤进行操作即可完成项目的启动和运行

编译所有文件

```
cd server
javac *.java
cd ..
cd client
javac *.java -classpath .:beautyeye_lnf.jar
cd ..
javac *.java
```

启动服务器

```
java ServerMain 8080
```

启动客户端

```
java -classpath .:client/beautyeye_lnf.jar ClientMain
```

GitHub

源码已经上传与GitHub:https://github.com/pudongqi/DesignPatternProject