<https://blog.csdn.net/u013064109/article/details/47843633>

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u013064109/article/details/47843633

聊天，QQ，微信，陌陌很多的即时通讯的软件，不管是桌面端还是移动端，在当今社交的时代都是不可或缺的一部分。这时候说Socket和ServerSocket感觉有点老调重弹感觉，相信很多人早就知道如何使用了，而且利用这个通信原理可能已经开发出很多优秀的通信软件吧，但是我感觉这个对于刚接触java网络编程的人来说，学会Socket通信实现聊天软件，是必须的一步，了解其中的原理更是非常重要的一步，对，很多人可能觉得对着视频敲出一个软件很容易，但是你能学到什么？？？盲目地崇拜大神吗？？，我认为需要花更多的时间去弄懂其实现的原理，然后总结一些属于自己的东西出来。本人菜鸟，但是喜欢分享一些自己的东西，希望能帮助需要帮助的人，不说废话，直接上...

为了照顾一下初学者下面就大概说下Socket的介绍:

1、Scoket又称“套接字”，应用程序通常通过“套接字”向网络发出请求或者应答网络请求  
在 java中Socket和ServerSocket类库位于java.net包中。ServerSocket用于服务器端  
Socket是建立网络连接时使用的，在连接成功时，应用程序两端都会产生一个Socket实例  
操作这个实例，完成所需的会话。对于一个网络连接来说，套接字是平等的，并没有差别  
不因为在服务器端或在客户端而产生不同的级别，不管是Socket还是ServerSocket他们的  
工作都是通过Socket类和其子类来完成的  
2、建立Socket链接可分三个步骤:  
         1.服务器监听  
         2.客户端发出请求  
         3.建立链接  
         4.通信  
3、Socket特点:  
          1.基于TCP链接，数据传输有保障  
          2.适用于建立长时间的链接，不像HTTP那样会随机关闭  
          3.Socket编程应用于即时通讯

4、ServerSocket的建立和使用:

public class ServerSocket\_Test {

 public static void main(String[] args) {  
  //port参数表示服务器监听的端口号，从1-65535  
  try {  
     
   ServerSocket serverSocket =new ServerSocket(12345);  
        
   //block，当没有客户端连接时，改主线程会一直阻塞等待连接，一直监听，直到有客户端连接才会执行  
  Socket socket= serverSocket.accept();//侦听事务的连接，accept是一个阻塞的方法，会阻塞当前的main线程，并且返回的是一个Socket类型  
  //建立连接，表示serverSocket在监听，如果监听到有客户端连接则会调用accept方法，然后返回一个Socket，最后建立连接  
  JOptionPane.showMessageDialog(null, "有客户端连接到了本机的12345端口");  
  //然后测试在浏览器中输入[http://127.0.0.1:12345](http://127.0.0.1:12345/)则会弹出相应有客户端连接的提示框，然后原来阻塞在accept方法那里就会往下执行  
  } catch (IOException e) {  
   e.printStackTrace();  
  }  
 }  
注意:但是以上的这种方法不推荐，因为这个里面有线程阻塞，会阻塞主线程，所以推荐一种更好的方法是就是单独开启一个线程去实现服务器监听客户端的连接

5、ServerSocketListener(单独的线程实现服务器的监听)

public class ServerListener extends Thread {  
 @Override  
 public void run() {  
  //port参数表示服务器监听的端口号，从1-65535  
  try {

   ServerSocket serverSocket =new ServerSocket(12345);

   while (true) {   //由于可能当有多个客户端连接时，accept方法就会产生多个Socket对象，需加一个while循环监听来自客户端的连接  
    //block，当没有客户端连接时，改主线程会一直阻塞等待连接，一直监听，直到有客户端连接才会执行  
    Socket socket= serverSocket.accept();//侦听事务的连接，accept是一个阻塞的方法，会阻塞当前的main线程，并且返回的是一个Socket类型  
    //建立连接，表示serverSocket在监听，如果监听到有客户端连接则会调用accept方法，然后返回一个Socket，最后建立连接  
    JOptionPane.showMessageDialog(null, "有客户端连接到了本机的12345端口");  
    //然后测试在浏览器中输入[http://127.0.0.1:12345](http://127.0.0.1:12345/)则会弹出相应有客户端连接的提示框，然后原来阻塞在accept方法那里就会往下执行  
     
    //将socket传递给另起的一个新的线程,即是socket通信的线程  
    new ChatSocket(socket).start();  
     
   }  
     
  } catch (IOException e) {  
   e.printStackTrace();  
  }  
 }

然后在主方法去开启这个线程即可:

public class ServerSocket\_Test {

 public static void main(String[] args) {  
  new ServerListener().start();  
 }

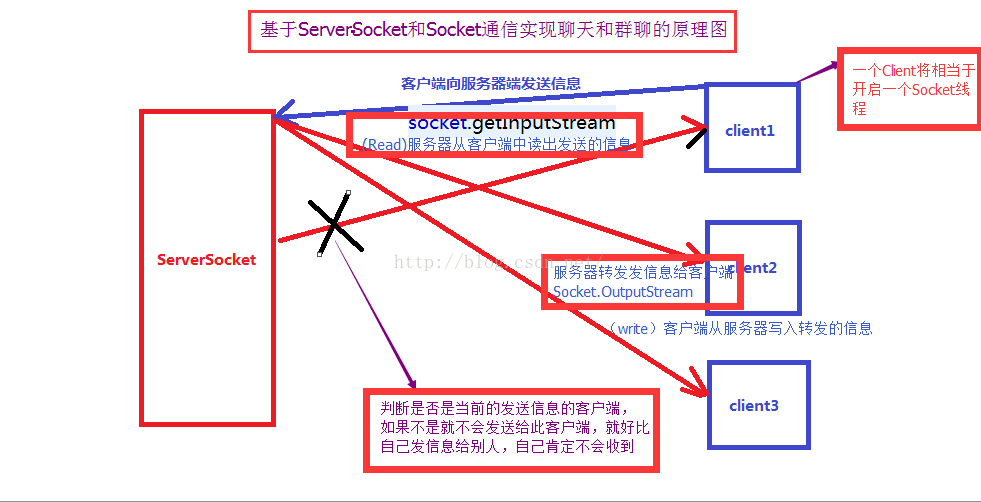
}

用浏览器打开运行结果（表示此时已经有客户端连接到服务器了，监听到客户端后就会弹出提示框，此时的浏览器就相当于客户端）：



那么接下来就讲解一下聊天服务器端（ServerSocket）的是实现:

主要实现原理:因为一个客户端就相当于开启一个Socket线程，然而要实现多人聊天，就相当于开启多个Socket线程（即多个客户端），然后把这些线程加入到Vector集合中去，当客户端（Socket）发送一条信息时，也就相当于服务器（ServerSocket）读入信息，而对于客户端是向服务器输入，实现输入流Socket.InputStream,然后利用BufferReader缓冲流读入服务器，然后在服务器（ServerSocket）中的去遍历这个线程集合，如果不是当前客户端对象就发送信息，这样就是实现了把当前客户端信息转发给其他的客户端使用Socket.OutputStream,即服务器向客户端输出流，并用PrintWriter流写入客户端。

具体见图:

ServerSocket代码:

1、ServerSocket.java：

public class ServerSocket {

 public static void main(String[] args) {  
  new ServerListener().start();  
  //运行的方法在command命令下输入:"telnet localhost 12345",每建立一个就是一个客户端，而且每个客户端享受不同的线程，等级是平等的  
 }

2.ServerListener.java

public class ServerListener extends Thread {  
 @Override  
 public void run() {  
  //port参数表示服务器监听的端口号，从1-65535  
  try {

   ServerSocket serverSocket =new ServerSocket(12345);  
   while (true) {//由于可能当有多个客户端连接时，accept方法就会产生多个Socket对象，需加一个while循环监听来自客户端的连接  
    Socket socket= serverSocket.accept();//侦听事务的连接，accept是一个阻塞的方法，会阻塞当前的main线程，并且返回的是一个Socket类型  
    //建立连接，表示serverSocket在监听，如果监听到有客户端连接则会调用accept方法，然后返回一个Socket，最后建立连接  
    JOptionPane.showMessageDialog(null, "有客户端连接到了本机的12345端口");  
      
    ChatSocket cs= new ChatSocket(socket);  
        cs.start();//开启ChatSocket线程  
        ChatManager.getchaChatManager().add(cs);  
   }  
     
  } catch (IOException e) {  
   e.printStackTrace();  
  }  
 }  
}  
3.ChatSocket.java

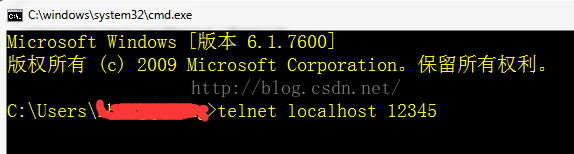
public class ChatSocket extends Thread {  
//创建一个Socket对象来接收SocketListener传来的Socket对象  
 Socket socket;  
 public ChatSocket(Socket s) {  
          this.socket=s;  
}  
 public void out(String out){  
  try {  
   socket.getOutputStream().write((out+"\n").getBytes("UTF-8"));//接收来自服务器端的数据  
  }catch (UnsupportedEncodingException e) {  
   e.printStackTrace();  
  }  
  catch (IOException e) {  
   System.out.println("断开了一个客户端链接");  
   ChatManager.getchaChatManager().remove(this);  
   e.printStackTrace();  
  }  
 }  
 @Override  
  public void run() {  
     out("您已经连接到服务器");  
    try {  
    
  BufferedReader br=new BufferedReader(  
    new InputStreamReader(  
      socket.getInputStream(),"UTF-8"));//当前服务器会不断读取当前客户端的数据  
  String line=null;  
     while ((line=br.readLine())!=null) {//客户端发送给服务器的数据  
    //然后服务器再将所有的信息转发给每一个客户端，调用publish方法  
        
      ChatManager.getchaChatManager().publish(this, line);  
   }  
     br.close();  
     System.out.println("断开了一个客户端链接");  
   ChatManager.getchaChatManager().remove(this);  
 } catch (IOException e) {  
  System.out.println("断开了一个客户端链接");  
  ChatManager.getchaChatManager().remove(this);  
  e.printStackTrace();  
 }  
       
 }

4、ChatManager.java

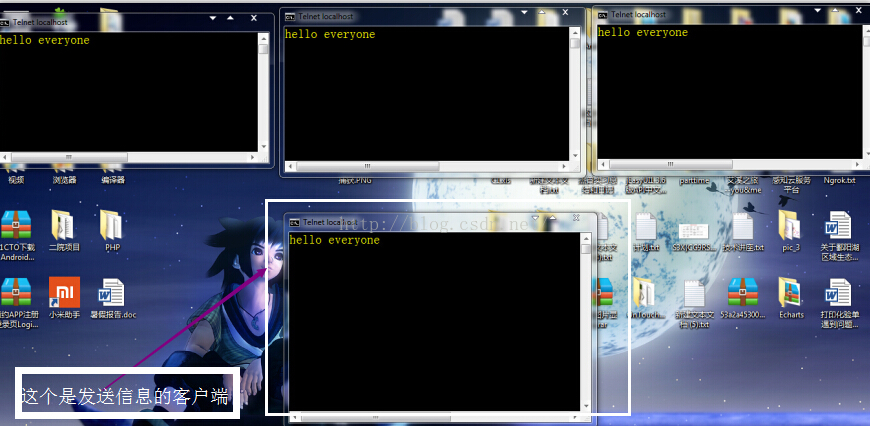
public class ChatManager {  
      //因为一个聊天服务器只有一个ChatManager所以需要创建一个单例  
   
 private ChatManager(){}  
 private static final ChatManager cm= new ChatManager();  
 public static  ChatManager getchaChatManager(){  
  return cm;  
 }  
   
 Vector<ChatSocket> vector=new Vector<ChatSocket>();  
 public void add(ChatSocket cs){  
      vector.add(cs);//将每一个线程加入集合中  
 }  
 public void remove(ChatSocket cs) {  
  vector.remove(cs);  
 }  
 public  void publish(ChatSocket cs,String chatinfo){//表示当前的线程给集合中的每一个线程发送的信息，也即当前的客户端给每一个客户端发送信息  
  //要给集合中所有的线程发送信息就必须遍历这个集合  
  for (int i = 0; i < vector.size(); i++) {  
   ChatSocket csChatSocket=vector.get(i);  
   if(!csChatSocket.equals(cs)){//则需要判断不许给当前客户端发送信息，也即不给自己发送信息  
    csChatSocket.out(chatinfo);//发送信息给其他的客户端  
   }  
  }  
 }  
}

到现在其实我们就可以测试聊天和多人聊天的功能:

运行的方法:我们打开Command命令，输入telnet localhost 12345，然后回车，就会建立起一个多人聊天室：



运行结果:



那接下来讲解聊天客户端的实现:

客户端的实现主要是有两部分，第一是GUI的实现，这个在这就不多讲了这是GUI（Swing）界面编程的知识，第二个就是客户端的逻辑的实现，因为客户端既要发送信息

又要接收信息，发送信息给服务器则需要Socket.getInputStream字节流，转换成InputStreamReader字符流，转换成BufferReader缓冲流，read到服务器；接收信息，从服务器中接收信息，则需要Socket.OutputStream字节流，转换成OutputStream转换成PrintWriter流，write到客户端。

代码:

ChatManager：

public class ChatManager {  
  private ChatManager(){}  
  private static final ChatManager instance =new ChatManager();  
  public static ChatManager getChatManager(){  
   return instance;  
  }  
  MainWindow window;  
  String IP;  
  Socket socket;  
  BufferedReader br;  
  PrintWriter pw;  
  public void setWindow(MainWindow window) {  
 this.window = window;  
 window.appendText("文本框已经和Manage绑定了");  
}  
  public void connect(String ip){  
   this.IP=ip;  
   new Thread(){

  @Override  
  public void run() {  
      try {  
       socket=new Socket(IP, 12345);//创建客户端，连接的端口是ServerSocket的端口  
       pw=new PrintWriter(  
         new OutputStreamWriter(  
           socket.getOutputStream(),"UTF-8") );  
       br=new BufferedReader(  
         new InputStreamReader(  
           socket.getInputStream(),"UTF-8"));  
       String line;  
         
       while ((line=br.readLine())!=null) {  
    window.appendText("收到:"+line);  
   }  
       br.close();  
      pw.close();  
      pw=null;  
      br=null;

      }catch (UnknownHostException e) {  
    e.printStackTrace();  
   }   
      catch (IOException e) {  
   e.printStackTrace();  
  }  
  }  
   }.start();  
  }  
  public void send(String out){  
   if(pw!=null){  
    pw.write(out+"\n");  
    pw.flush();  
   }else{  
  window.appendText("已中断与服务器的连接");    
   }  
  }  
}

StartClient.java：

public class StartClient {

 public static void main(String[] args) {  
  EventQueue.invokeLater(new Runnable() {  
   public void run() {  
    try {  
     MainWindow frame = new MainWindow();  
     frame.setVisible(true);  
     ChatManager.getChatManager().setWindow(frame);  
    } catch (Exception e) {  
     e.printStackTrace();  
    }  
   }  
  });  
 }

}

GUI界面代码:

package com.zhongqihong.client.view;

import java.awt.BorderLayout;  
import java.awt.EventQueue;

import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JTextArea;  
import javax.swing.GroupLayout;  
import javax.swing.GroupLayout.Alignment;  
import javax.swing.JTextField;  
import javax.swing.JButton;  
import javax.swing.LayoutStyle.ComponentPlacement;

import com.zhongqihong.client.ChatManager;

import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;

public class MainWindow extends JFrame {

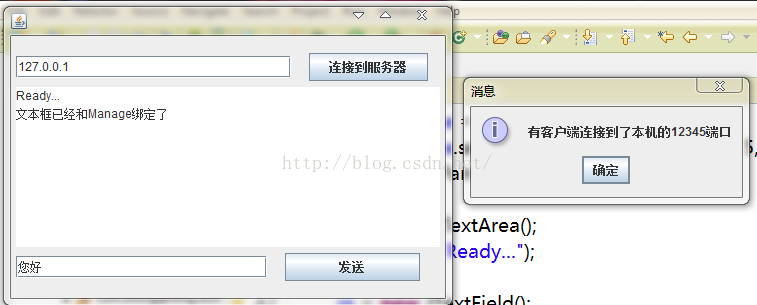
 private static final long serialVersionUID = 1L;  
 private JPanel contentPane;  
 JTextArea txt;  
 private JTextField ip;  
 private JTextField send;  
 /\*\*  
  \* Create the frame.  
  \*/  
 public MainWindow() {  
  setAlwaysOnTop(true);  
  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  setBounds(100, 100, 450, 300);  
  contentPane = new JPanel();  
  contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
  setContentPane(contentPane);  
    
  txt = new JTextArea();  
  txt.setText("Ready...");  
    
  ip = new JTextField();  
  ip.setText("127.0.0.1:808");  
  ip.setColumns(10);  
    
  JButton button = new JButton("\u8FDE\u63A5\u5230\u670D\u52A1\u5668");  
  button.addMouseListener(new MouseAdapter() {  
   @Override  
   public void mouseClicked(MouseEvent e) {  
    ChatManager.getChatManager().connect(ip.getText());  
   }  
  });  
    
  send = new JTextField();  
  send.setText("\u60A8\u597D");  
  send.setColumns(10);  
    
  JButton button\_1 = new JButton("\u53D1\u9001");  
  button\_1.addMouseListener(new MouseAdapter() {  
   @Override  
   public void mouseClicked(MouseEvent e) {  
    ChatManager.getChatManager().send(send.getText());  
     appendText("我说:"+send.getText());  
        send.setText("");  
   }  
  });  
  GroupLayout gl\_contentPane = new GroupLayout(contentPane);  
  gl\_contentPane.setHorizontalGroup(  
   gl\_contentPane.createParallelGroup(Alignment.LEADING)  
    .addGroup(Alignment.TRAILING, gl\_contentPane.createSequentialGroup()  
     .addComponent(ip, GroupLayout.DEFAULT\_SIZE, 277, Short.MAX\_VALUE)  
     .addGap(18)  
     .addComponent(button, GroupLayout.PREFERRED\_SIZE, 119, GroupLayout.PREFERRED\_SIZE)  
     .addContainerGap())  
    .addGroup(Alignment.TRAILING, gl\_contentPane.createSequentialGroup()  
     .addComponent(send, GroupLayout.DEFAULT\_SIZE, 251, Short.MAX\_VALUE)  
     .addGap(18)  
     .addComponent(button\_1, GroupLayout.PREFERRED\_SIZE, 135, GroupLayout.PREFERRED\_SIZE)  
     .addGap(20))  
    .addComponent(txt, GroupLayout.DEFAULT\_SIZE, 424, Short.MAX\_VALUE)  
  );  
  gl\_contentPane.setVerticalGroup(  
   gl\_contentPane.createParallelGroup(Alignment.LEADING)  
    .addGroup(gl\_contentPane.createSequentialGroup()  
     .addContainerGap()  
     .addGroup(gl\_contentPane.createParallelGroup(Alignment.BASELINE)  
      .addComponent(ip, GroupLayout.PREFERRED\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)  
      .addComponent(button))  
     .addPreferredGap(ComponentPlacement.RELATED)  
     .addComponent(txt, GroupLayout.DEFAULT\_SIZE, 174, Short.MAX\_VALUE)  
     .addPreferredGap(ComponentPlacement.RELATED)  
     .addGroup(gl\_contentPane.createParallelGroup(Alignment.BASELINE)  
      .addComponent(send, GroupLayout.PREFERRED\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)  
      .addComponent(button\_1))  
     .addContainerGap())  
  );  
  contentPane.setLayout(gl\_contentPane);  
 }  
public void appendText(String in){  
 txt.append("\n"+in);  
}  
}

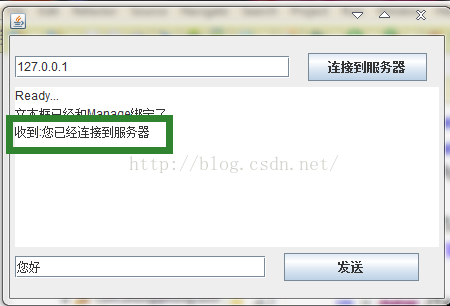
public class ServerSocket\_Test {

 public static void main(String[] args) {  
  new ServerListener().start();  
  //运行的方法在command命令下输入:"telnet localhost 12345",每建立一个就是一个客户端，而且每个客户端享受不同的线程，等级是平等的  
 }

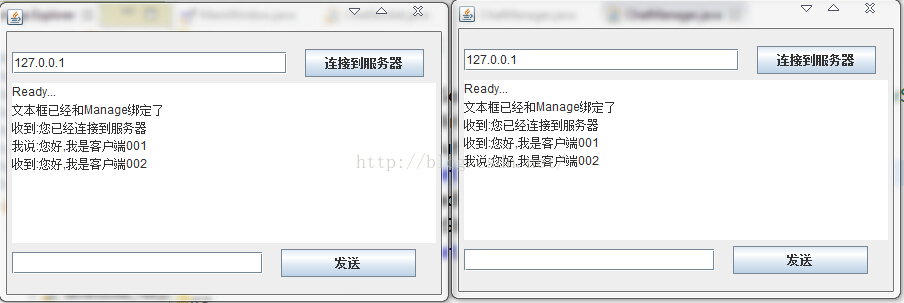
}

 }  
 }  
  
运行代码:

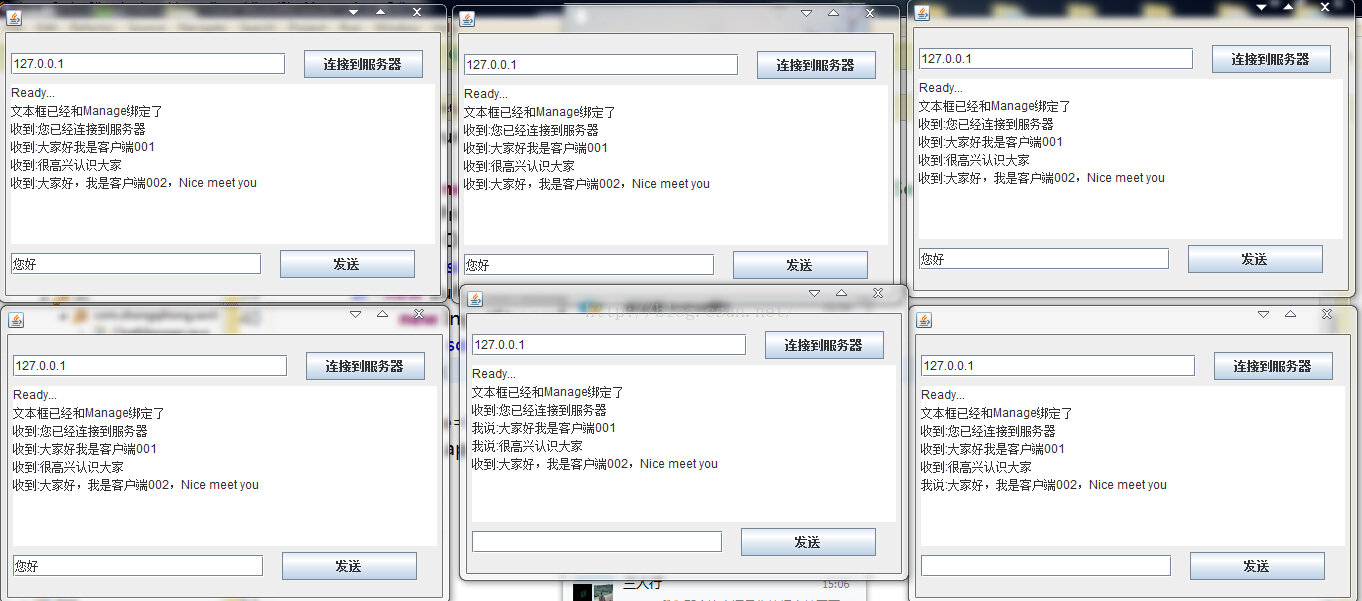




两人私聊:



多人群聊:



到这里，我们的多人聊天的客户端就成功实现了，其实一步一步来，把复杂的问题分解成一个个小问题来解决就可以了。

PS：这里是Demo的源码:http://pan.baidu.com/s/1qWQikMC