第4章 分布式软件体系结构风格

4.2 二层C/S体系结构

2-Tier C/S Architecture

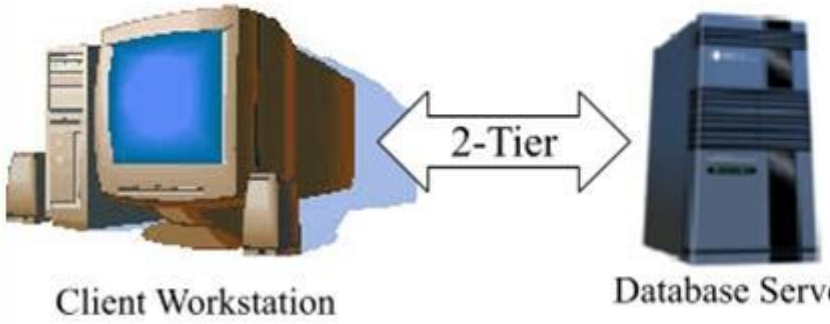
刘其成

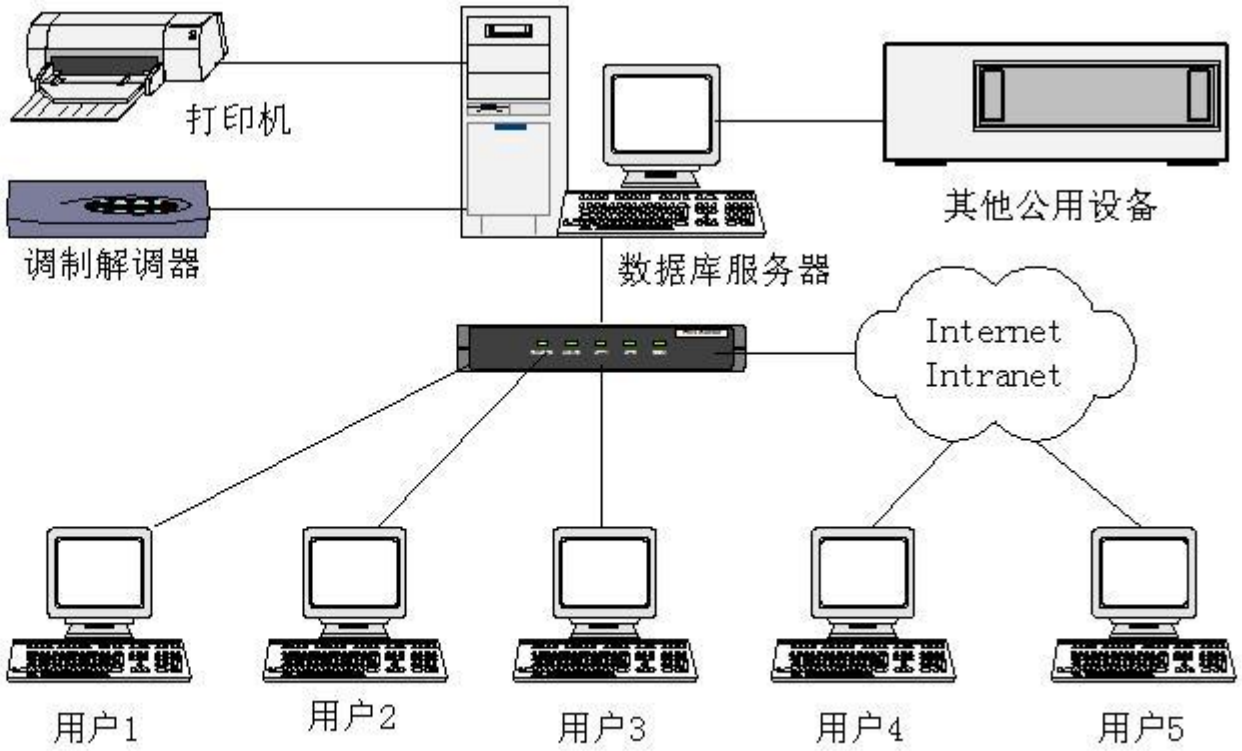
计算机与控制工程学院

ytliuqc@163.com

2018-09

两层C/S结构 (2-Tier Client/Server)





两层C/S结构 (2-Tier Client/Server)

- 两层C/S结构
  - 用户界面处于客户机
  - 数据库管理服务处于服务器端，通常是存储过程/触发器的形式
  - 业务处理过程—即业务逻辑—被分解为客户机与服务器两部分
- 两层C/S 体系结构将应用一分为二，服务器(后台)负责数据管理，客户机(前台)完成与用户的交互任务。
- 服务器为多个客户应用程序管理数据，而客户程序发送、请求和分析从服务器接收的数据，这是一种“胖客户机”、“瘦服务器”的体系结构。

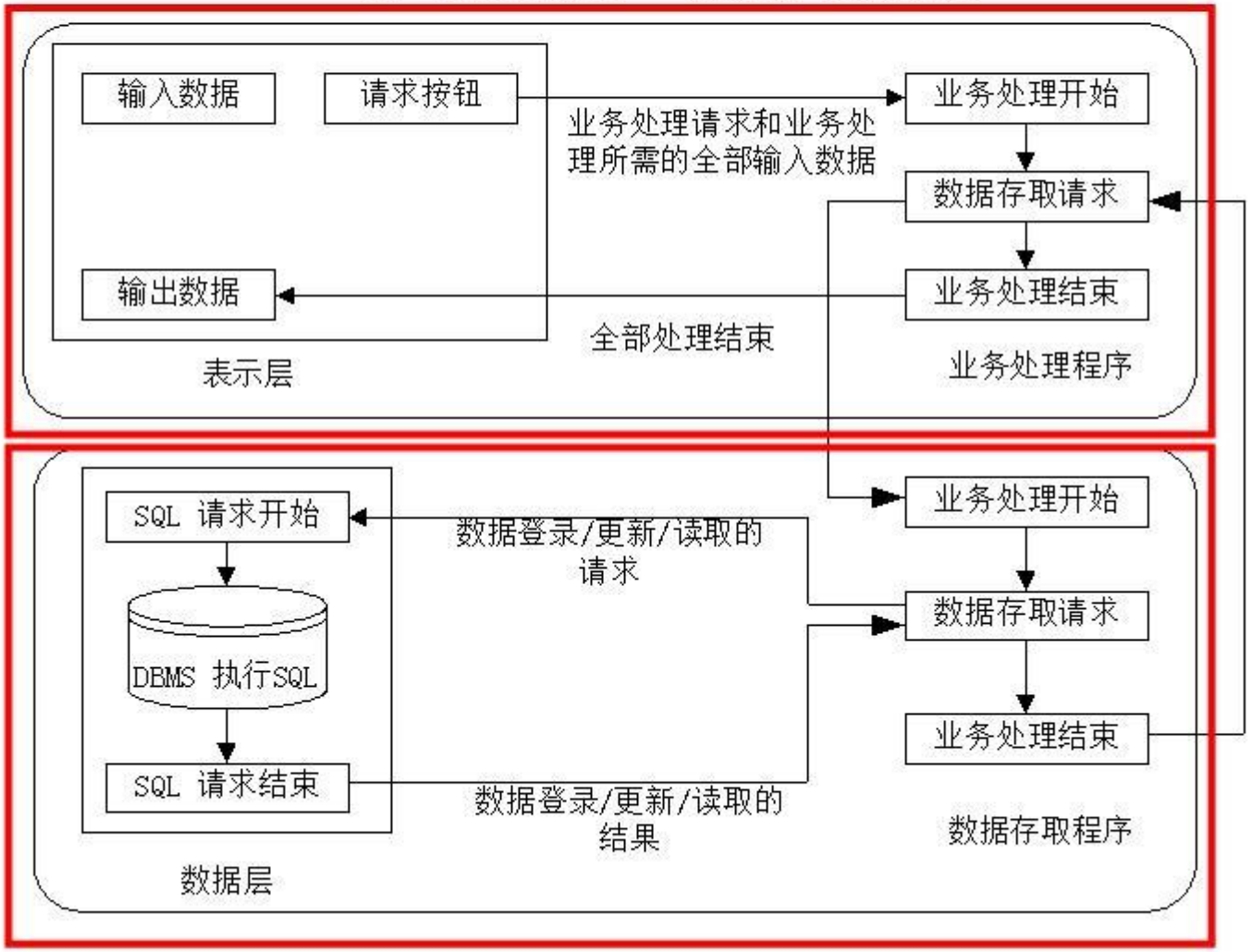
两层C/S结构 (2-Tier Client/Server)

- 基本组件：
  - 数据库服务器
    - 存放数据的数据库、负责数据处理的业务逻辑；
  - 客户机应用程序：
    - GUI：用户界面
    - 业务逻辑：利用客户机上的应用程序对数据进行处理；
- 连接件：经由网络的调用-返回机制或隐式调用机制。
  - 客户机←→服务器：客户机向服务器发送请求；服务器进行相关处理，将结果发给客户端；客户端接收返回结果。

- 服务器负责有效地管理系统的资源，其任务集中于：
  - 数据库安全性的要求。
  - 数据库访问并发性的控制。
  - 数据库前端的客户应用程序的全局数据完整性规则。
  - 数据库的备份与恢复。
- 客户应用程序的主要任务是：
  - 提供用户与数据库交互的界面。
  - 向数据库服务器提交用户请求，并接收来自数据库服务器的信息
  - 利用客户应用程序对存在于客户端的数据执行应用逻辑要求
- 网络通信软件的主要作用是完成数据库服务器和客户应用程序之间的数据传输。

两层C/S结构 (2-Tier Client/Server)

C/S结构的一般处理流程



两层C/S体系结构优点

- 两层C/S 体系结构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受。
- 系统的客户应用程序和服务器组件分别运行在不同的计算机上，系统中每台服务器都可以适合各组件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。
- 在两层C/S体系结构中，系统中的功能组件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理。将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。

两层C/S体系结构缺点

- 两层C/S体系结构虽然具有一些优点，但随着企业规模的日益扩大，软件的复杂程度不断提高，两层C/S体系结构逐渐暴露出以下缺点：
  - 互操作性差：
    - 采用不同开发工具或平台开发的软件，一般互不兼容，不能或很难移植到其他平台上运行。使用DBMS所提供的私有的数据编程语言来开发业务逻辑，降低了DBMS选择的灵活性
    - ——导致：软件移植困难，新技术无法轻易使用
      - 例如：Oracle DB提供的若干存储过程函数，在SQL Server上就不能执行

两层C/S体系结构缺点

- 系统管理与配置成本高：
  - 采用两层C/S体系结构的软件要升级，开发人员必须到现场为客户机升级，每个客户机上的软件都需维护。当系统升级时，对软件的一个小小改动，每一个客户端必须更新。
  - ——导致：软件维护和升级困难
- 系统伸缩性差：
  - 当用户数超过100，性能急剧恶化
  - 服务器成为系统的瓶颈

两层C/S体系结构缺点

- 开发成本较高。
  - 两层C/S体系结构对客户端软硬件配置要求较高，尤其是软件的不断升级，对硬件要求不断提高，增加了整个系统的成本，且客户端变得越来越臃肿。
- 客户端程序设计复杂。
  - 采用两层C/S体系结构进行软件开发，大部分工作量放在客户端的程序设计上，客户端显得十分庞大。
- 用户界面风格不一，使用繁杂，不利于推广使用。

适用场合

- 两层C/S架构通常被用在那些管理与操作不太复杂的非实时的信息处理系统
- 适合于轻量级事务——客户机对服务器的请求少，数据传输量少
- 当业务逻辑较少变化以及用户数少于100时，两层C/S架构的性能较好

1. 一个简单的Client/Server程序

- 这里举一个简单的例子说明如何运用套接字对服务器和客户机进行操作。客户机发送数据到服务器，服务器将收到的数据返回给客户机。
- 服务器的全部工作就是等候建立一个连接，然后用这个连接产生的socket创建一个输入流和一个输出流，它把从输入流中读入的数据都反馈给输出流。直到接收到字符串“end”为止，最后关闭连接。
- 客户机连接服务器，然后创建一个输入流和一个输出流，它把数据通过输出流传给服务器，然后从输入流中接收服务器发回的数据。

（1）服务器端程序tcpServer.java

```
import java.io.*;
import java.net.*;
public class tcpServer {
    public static final int PORT=8888;
    public static void main(String[] args) throws IOException{
          //建立ServerSocket
        ServerSocket s=new ServerSocket(PORT);
        System.out.println("ServerSocket:"+s);
        try{  /*程序阻塞,等待连接。
                即直到有一个客户请求到达,程序方能继续执行*/
            Socket ss=s.accept();
            System.out.println("Socket accept:"+ss);
            try {
```

```
            //连接成功，建立相应的I/O数据流
            DataInputStream dis=new  DataInputStream(ss.getInputStream());
            DataOutputStream dos=new DataOutputStream(
                                                    ss.getOutputStream());
          while(true){//在循环中，与客户机通信
              String str=dis.readUTF();                  //从客户机中读数据
              if(str.equals("end"))break;                //当读到end时，程序终止
              System.out.println(str);
              dos.writeUTF("Echoing:"+str);     //向客户机中写数据
            }
            dos.close();
            dis.close();
          }finally{
            ss.close();
          }
        }finally{
          s.close();
        }
    }
}
```

- 服务器端运行结果为：
  - ServerSocket:ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=8888]
  - Socket accept:Socket[addr=/127.0.0.1,port=7312,localport=8888]
  - 测试:0
  - 测试:1
  - 测试:2
  - 测试:3
  - 测试:4
  - 测试:5

（2）客户机端程序tcpClient.java

```
import java.io.*;
import java.net.*;
public class tcpClient {
  public static void main(String[] args) throws IOException{
    //建立Socket，服务器在本机的8888端口处进行“侦听”
    Socket ss=new Socket("127.0.0.1",8888);
    System.out.println("Socket:"+ss);
    try{
      //套接字建立成功，建立I/O流进行通信
      DataInputStream dis=new DataInputStream(
                                          ss.getInputStream());
      DataOutputStream dos=new DataOutputStream(
                                          ss.getOutputStream());
```

```
        for(int i=0;i<6;i++){
          dos.writeUTF("测试:"+i);      //向服务器发数据
          dos.flush();   //刷新输出缓冲区，以便立即发送
          System.out.println(dis.readUTF());
                    //将从服务器接收的数据输出
        }
        dos.writeUTF("end");   //向服务器发送终止标志
        dos.flush();      //刷新输出缓冲区，以便立即发送
        dos.close();
        dis.close();
      }finally{
        ss.close();
      }
    }
}
```

- 客户器端运行结果为：
  - Socket:Socket[addr=/127.0.0.1,port=8888,localport=7312]
  - Echoing:测试:0
  - Echoing:测试:1
  - Echoing:测试:2
  - Echoing:测试:3
  - Echoing:测试:4
  - Echoing:测试:5
- 可以看到，关闭套接字的语句放在finally块中，这是为了确保一定能够关闭socket
- 但关闭I/O流的语句没有放在finally块中，这是因为在try块内，可能还没有建立I/O流对象就抛出异常。

- 组件
  - tcpServer类
  - tcpClient类

连接件

- ①建立网络连接
  - 服务端创建ServerSocket类的对象 ServerSocket s=new ServerSocket(PORT), PORT是服务对应的端口号。然后，Socket ss=s.accept()，ServerSocket 类的对象s调用accept()方法，返回一个Socket 类的对象ss，等待客户机发送请求；如果没有客户机端发送请求，则程序阻塞。
  - 在客户端创建一个Socket 的对象ss，Socket ss=new Socket("127.0.0.1",8888)，第一个参数是要连接的服务器的IP地址，第二个参数是提供的端口号，作用是向服务器端发送请求，同时将请求送给了服务器程序中的accept()的方法。
  - 服务器中ServerSocket的accept()方法接收到客户机的网络连接请求，会把该请求送给服务器的Socket类的对象ss。这时，客户机和服务器建立起了连接。
- ②客户机与服务器建立I/O数据流进行通信
  - 在客户机端，通过dos.writeUTF()，向服务器发送数据即一个整型数组，在服务端通过dis.readInt()来接收数组，并对数据进行相应的处理，数据的结果通过dos.writeUTF()返回给客户端，客户端通过dis.readUTF()接收服务器返回回来的数据，并输出数据。

2. 多用户机制

- 在上例中，服务器每次只能为一个客户提供服务。但是，一般情况下要求服务器能同时处理多个客户机的请求。
  - 解决这个问题的关键就是多线程处理机制
- 常用的方法
  - 在服务器程序里，在accept()返回一个socket后，就用它新建一个线程，令它只为那个特定的客户服务。
  - 然后再调用accept()，等待下一次新的连接请求。
- 下例是一个服务多个客户的网络程序。
  - 它与上例很相似，只是为一个特定的客户提供服务的所有操作都已移入一个独立的线程类中。

（1）服务器端程序MTcpServer.java

```
import java.net.*;
import java.io.*;
public class MTcpServer {
public static final int PORT=8888;
    public static void main(String[] args) throws IOException{
        ServerSocket ss=new ServerSocket(PORT);
        System.out.println("服务器开启");
        try{
            while(true){
                Socket s=ss.accept();
                try{//启动一个新的线程，并把accept()得到的socket传入新线程中
                    new ServerOne(s);
                }catch(IOException e){
                    s.close();
                }
            }
        }finally{
            ss.close();
        }
    }
}
```

```
class ServerOne extends Thread{//用于为特定用户服务的线程类
  private Socket s;
  private BufferedReader in;
  private PrintWriter out;

  public ServerOne(Socket s) throws IOException{
    this.s=s;
    in=new BufferedReader(new InputStreamReader(
                                        s.getInputStream()));
    out=new PrintWriter(new BufferedWriter(new
                    OutputStreamWriter(s.getOutputStream())),true);
    start();
  }
```

```
    public void run(){
        try{
            while(true){
                String str=in.readLine();
                if(str.equals("end")) break;
                System.out.println(str);
                out.println("Echo:"+str);
            }
          System.out.println("closing.....");
        }catch(IOException e){
        }finally{
            try{
                s.close();
          }catch(IOException e){}
        }
    }
}
```

- 为了证实服务器代码确实能为多名客户提供服务，下面这个程序将创建多个客户（使用线程），并同相同的服务器建立连接。

（2）客户端程序MTcpClient.java

```
import java.net.*;
import java.io.*;
public class MTcpClient extends Thread{
    //允许创建的线程的最大数
    static final int MAX_THREADS=30;
    private Socket s;
    private BufferedReader in;
    private PrintWriter out;
    //每一个线程的id都不同。
    private static int id=0;
    //当前活动的线程数
    private static int threadCount=0;
    public static int getThreadCount(){
        return threadCount;
    }
```

```
public MTcpClient(InetAddress ia) {
    System.out.println("Making client"+id);
    threadCount++;
    id++;
    try{
        s=new Socket(ia,MTcpServer.PORT);
    }catch(IOException e){}
    try{
        in=new BufferedReader(new InputStreamReader
                                    (s.getInputStream()));
         out=new PrintWriter(new BufferedWriter (new
                  OutputStreamWriter(s.getOutputStream())),true);
         start();
    }catch(IOException e1){
        try{
             s.close();
        }catch(IOException e2){}
    }
}
```

```
public void run(){
    try{
        String str;
        for(int i=0;i<25;i++){
            out.println("Client #"+id+":"+i);
            str=in.readLine();
            System.out.println(str);
        }
        out.println("end");
    }catch(IOException e){
    }finally{
        try{
            s.close();
        }catch(IOException e){}
        threadCount--;
    }
}
```

```
public static void main(String[] args)throws
                              IOException,InterruptedException{
    InetAddress ia=InetAddress.getByName(null);
    while(true){
      if(getThreadCount()<MAX_THREADS)
                new MTcpClient(ia);
      Thread.currentThread().sleep(10);
    }
  }
}
```

3. 数据库访问

- 本例利用套接字技术，实现应用程序中，对数据库的访问。
- 应用程序只是利用套接字连接，向服务器发送一个查询的条件；而服务器负责对数据库的查询，然后服务器再将查询的结果，利用建立的套接字，返回给客户端。

（1）服务器端程序Server.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.sql.*;
public class Server{
   public static void main(String args[]){
     Connection con;
     PreparedStatement sql=null;
     ResultSet rs;
     try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     }catch(ClassNotFoundException e){}
     try{
        con=DriverManager.getConnection("jdbc:odbc:myDB","liu","123");
        sql=con.prepareStatement(
                          "SELECT * FROM chengjibiao WHERE number = ? ");
     }catch(SQLException e){
         System.out.println(e);
     }
```

```
        ServerSocket server=null;
        Server_thread thread;
        Socket you=null;
        while(true){
          try{
              server=new ServerSocket(4331);
          }catch(IOException e1){
              System.out.println("正在监听");
          }
          try{
              System.out.println(" 等待客户呼叫");
              you=server.accept();
              System.out.println("客户的地址:"+you.getInetAddress());
         }catch(IOException e1){
              System.out.println("正在等待客户");
          }
          if(you!=null){
              new Server_thread(you,sql).start();
          }
        }
      }
  }
```

```
class Server_thread extends Thread{
    Socket socket;
    DataOutputStream out=null;
    DataInputStream  in=null;
    PreparedStatement sql;
    boolean boo=false;
    Server_thread(Socket t, PreparedStatement sql){
      socket=t;
      this.sql=sql;
      try {
        out=new DataOutputStream(socket.getOutputStream());
        in=new DataInputStream(socket.getInputStream());
      }catch(IOException e){}
    }
```

```
public void run(){
while(true){
try{
    String num=in.readUTF();
    boo=false;
    sql.setString(1,num);
    ResultSet rs=sql.executeQuery() ;
    while(rs.next()){
       boo=true;
       String number=rs.getString(1);
       String name=rs.getString(2);
       String date=rs.getString(3);
       int math=rs.getInt(4);
       int english=rs.getInt(5);
       out.writeUTF("学号:"+number+" 姓名:"+name+" 出生:“
                    +date+" 数学:"+math+" 英语"+english);
    }
```

```
            if(boo==false){
                out.writeUTF("没有该学号！ ");
            }
        }catch (Exception e){
            System.out.println("客户离开"+e);
            return;
        }
    }
  }
}
```

（2） 客户端程序Client.java

```
import java.net.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Client{
    public static void main(String args[]){
        new QueryClient();
    }
}
```

```
class QueryClient extends Frame implements Runnable, ActionListener{
    Button connection,send;
    TextField inputText;
    TextArea showResult;
    Socket socket=null;
    DataInputStream in=null;
    DataOutputStream out=null;
    Thread thread;
```

```
QueryClient(){
  socket=new Socket();
  Panel p=new Panel();
  connection=new Button("连接服务器");
  send=new Button("发送");
  send.setEnabled(false);
  inputText=new TextField(8);
  showResult=new TextArea(6,42);
  p.add(connection);
  p.add(new Label("输入学号"));
  p.add(inputText);
  p.add(send);
  add(p,BorderLayout.NORTH);
  add(showResult,BorderLayout.CENTER);
  connection.addActionListener(this);
  send.addActionListener(this);
  thread=new Thread(this);
  setBounds(10,30,350,400);
  setVisible(true);
  validate();
  addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
      System.exit(0);
    }
  });
}
```

```
public void actionPerformed(ActionEvent e){
  if(e.getSource()==connection){
    try{
      if(socket.isConnected()){}
      else{
        InetAddress address=InetAddress.getByName("127.0.0.1");
        InetSocketAddress socketAddress=new
                                        InetSocketAddress(address,4331);
        socket.connect(socketAddress);
        in = new DataInputStream(socket.getInputStream());
        out = new DataOutputStream(socket.getOutputStream());
        send.setEnabled(true);
        thread.start();
      }
    }catch (IOException ee){}
  }
  if(e.getSource()==send){
    String s=inputText.getText();
    if(s!=null){
      try {
        out.writeUTF(s);
      }catch(IOException e1){
      }
    }
  }
}
```

```
public void run(){
  String s=null;
  while(true){
    try{
      s=in.readUTF();
      showResult.append("\n"+s);
    }catch(IOException e1){
      showResult.setText("与服务器已断开");
      break;
    }
  }
}
```

思考题

- 二层C/S体系结构：组件、连接件、工作机制、特点。相关程序，语言不限。

谢谢

2018年10月29日