* 学习目标

  * 能够掌握线程池

      * ThreadPoolExecutor,封装ThreadPoolUtils

  * 能够理解网络编程的概述

      * 网络设备之间通讯

  * 能够掌握网络编程的三要素

      * 协议，IP，端口

      * ISO，TCP/IP4,5

      * TCP:面向连接，可靠，安全，基于字节流

          * 三次握手的机制

       * UDP

  * 能够掌握TCP通讯的概述

  * 能够掌握Socket和ServerSocket类

  * 能够掌握网络编程HelloWorld开发

  * 能够掌握网络编程案例之文件上传

  * 能够掌握网络编程案例之编写简易版tomcat

---

  * 回顾

    * BlockingQueue:FIFO,阻塞特点，继承Queue继承Collection

        * add offer put take drainTo,....

        * ArrayBlockingQueue，LinkedBlockingQueue，PriorityBlockingQueue(Comparable,Comparator)

        * DelayQueue (Delay) ,SynchronousQueue

    * 线程池

      * 常量池，数据库连接池 ，线程池--性能优化技术

* Executor--->execute(Runnable)

* OOP六大设计原则：接口隔离原则，开闭原则，依赖倒置, 最少知识，组合和聚合， 里式替换

* BlockingQueue queue=new ArrayBlockingQueue();

* ExecutorService:线程池管理

* 常见的方法

* shutdown(不接受新的任务，会执行队列任务)

* shutdownNow(不接受新的任务，不执行队列任务和尝试终止正在执行任务)

* Future sumbit(Callable-call)

* Executors

* newSingleThreadExecutor();

* newFixedThreadPool(int n);

* newCachedThreadPool();

* newScheduledThreadPool(int n);

* 延迟执行某个任务，周期性执行某个任务

* 一个周期时间不受任务执行时间影响，...

* ....

* ThreadPoolExecutor

* 线程池状态：Running---> ShutDown -->Stop --> Tyding ---Terminated

* 线程构建的参数

* corePoolSize：核心线程数（师徒5人）

* maxPoolSize: 临时线程数+核心线程数=池最大线程数

* keepAlivedTime:临时线程空闲的存活时间

* TimeUnit

* blockingQueue

* ThreadFactory

* RejectedExecutionHandler

* AbortPolicy

* CallerPolicy

* DiscardPolicy

* DiscardOldestPolicy

* 能够掌握线程池

* 能够理解网络编程的概述

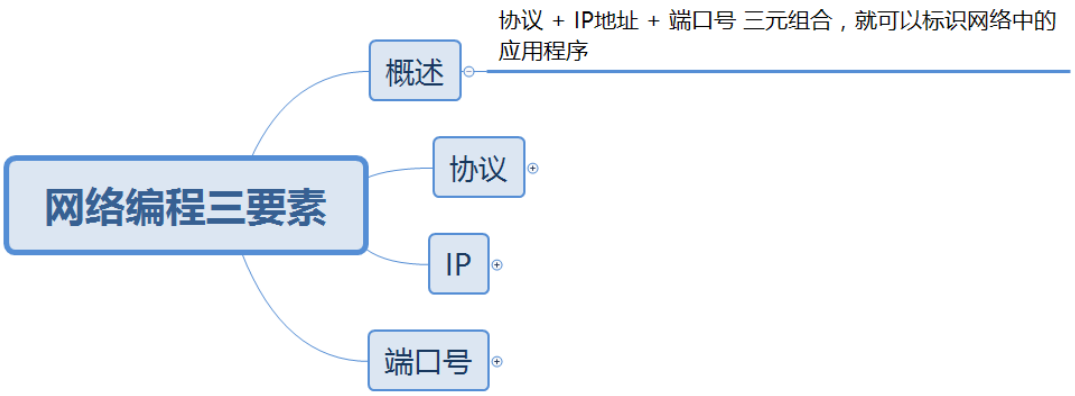
* B/S与C/S

* 参考：01-html中的B/S与C/S

* 网络编程，就是在一定的协议下，实现两台计算机的通信的程序。

* 网络编程最主要的工作就是在发送端把信息通过规定好的协议进行组装包，在接收端按照规定好的协议把包进行解析，从而提取出对应的信息，达到通信的目的。

* 能够掌握网络编程的三要素



* 协议

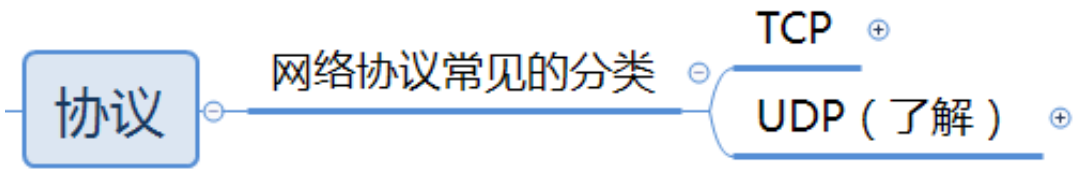
* 网络通信协议是一种网络通用语言，为连接不同操作系统和不同硬件体系结构的互联网络提供通信支持，是一种网络通用语言。

* OSI是一个开放性的通信系统互连参考模型，它是一个定义得非常好的协议规范。

* OSI七层协议，TCP/IP五层协议,TCP/IP四层协议

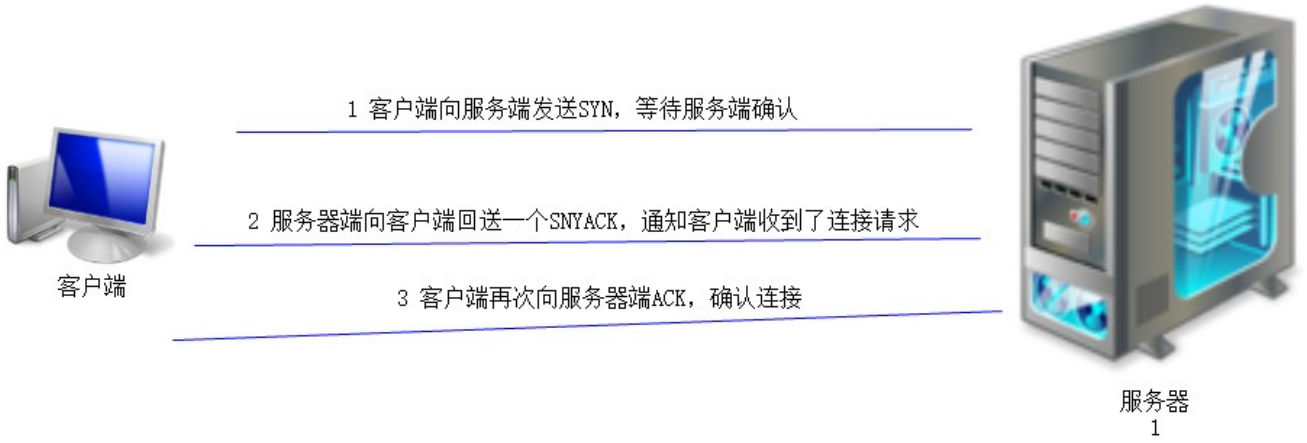

OSI协议
应用层-例HTTP
表示层-数据格式及加密（http协议）
会话层-例SQL，RPC
传输层-例TCP，UDP
网络层-例IP
数据链路层-例wifi
物理层-例光纤
TCP/IP五层
应用层
传输层-例TCP，UDP
网络层-例IP
数据链路层-例wifi
物理层-例光纤
TCP/IP四层
应用层
传输层-例TCP，UDP
网络层-例IP
网络接口层

* 常见协议的分类

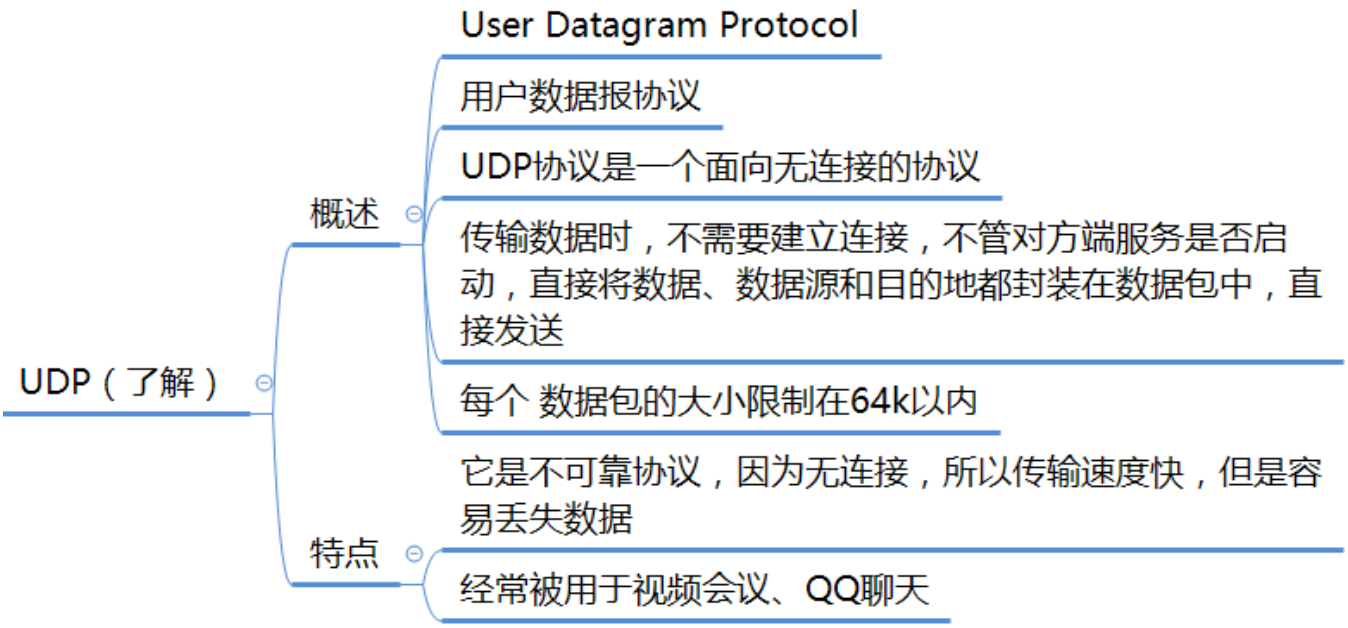

协议
网络协议常见的分类
TCP
UDP（了解）

* TCP

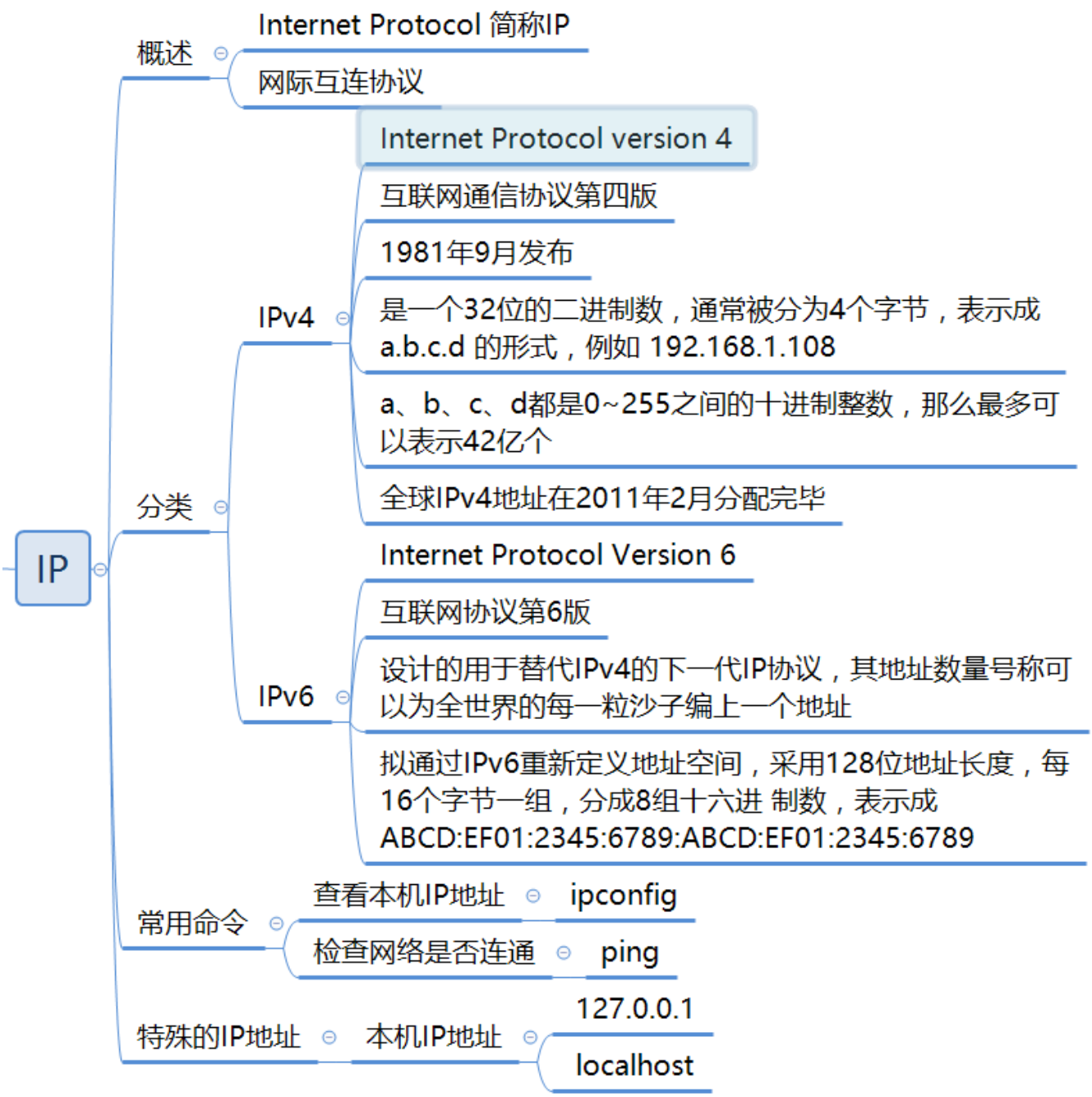
- TCP
  - 概述
    - Transmission Control Protocol
    - 传输控制协议
    - 1981年9月定义的
    - 是一种面向连接的、可靠的、基于字节流的传输层通信协议
  - 三次握手
    - TCP协议中，在发送数据的准备阶段，客户端与服务器之间的三次交互，以保证连接的可 靠
    - 第一次握手：客户端向服务器端发出连接请求，等待服务器确认
    - 第二次握手：服务器端向客户端回送一个响应，通知客户端收到了连接请求
    - 第三次握手：客户端再次向服务器端发送确认信息，确认连接
    - 完成三次握手，连接建立后，客户端和服务器就可以开始进行数据传输了
  - 总结
    - TCP协议可 以保证传输数据的安全，所以应用十分广泛，例如下载文件、浏览网页等。



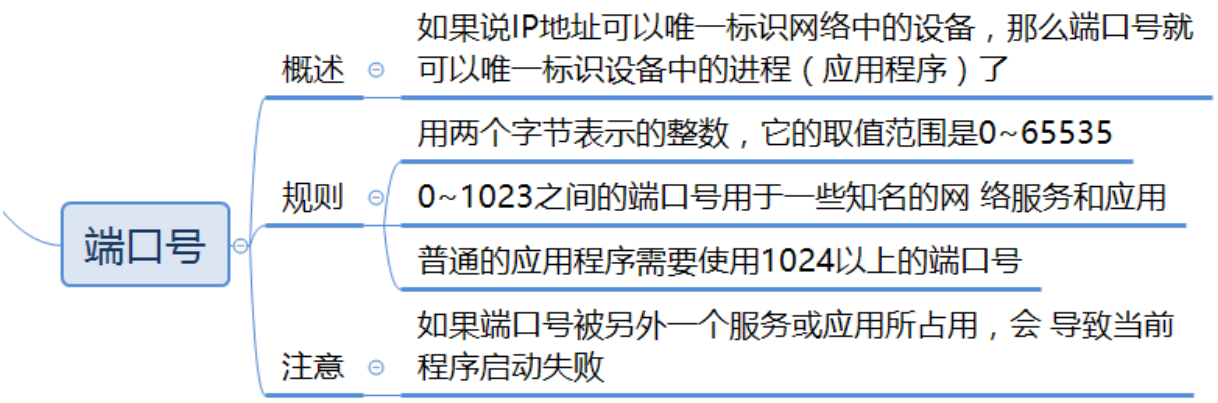
* UDP


UDP（了解）
概述
User Datagram Protocol
用户数据报协议
UDP协议是一个面向无连接的协议
传输数据时，不需要建立连接，不管对方端服务是否启动，直接将数据、数据源和目的地都封装在数据包中，直接发送
每个 数据包的大小限制在64k以内
特点
它是不可靠协议，因为无连接，所以传输速度快，但是容易丢失数据
经常被用于视频会议、QQ聊天

* IP地址


IP
概述
Internet Protocol 简称IP
网际互连协议
分类
IPv4
Internet Protocol version 4
互联网通信协议第四版
1981年9月发布
是一个32位的二进制数，通常被分为4个字节，表示成a.b.c.d 的形式，例如 192.168.1.108
a、b、c、d都是0~255之间的十进制整数，那么最多可以表示42亿个
全球IPv4地址在2011年2月分配完毕
IPv6
Internet Protocol Version 6
互联网协议第6版
设计的用于替代IPv4的下一代IP协议，其地址数量号称可以为全世界的每一粒沙子编上一个地址
拟通过IPv6重新定义地址空间，采用128位地址长度，每16个字节一组，分成8组十六进 制数，表示成ABCD:EF01:2345:6789:ABCD:EF01:2345:6789
常用命令
查看本机IP地址
ipconfig
检查网络是否连通
ping
特殊的IP地址
本机IP地址
127.0.0.1
localhost

* 端口号


端口号
概述
如果说IP地址可以唯一标识网络中的设备，那么端口号就可以唯一标识设备中的进程（应用程序）了
规则
用两个字节表示的整数，它的取值范围是0~65535
0~1023之间的端口号用于一些知名的网 络服务和应用
普通的应用程序需要使用1024以上的端口号
注意
如果端口号被另外一个服务或应用所占用，会 导致当前程序启动失败

* 能够掌握TCP通讯的概述

* TCP通信能实现两台计算机之间的数据交互，通信的两端，要严格区分为客户端（Client）与服务端（Server）

* 两端通信时步骤

* 服务端程序，需要事先启动，等待客户端的连接

* 客户端主动连接服务器端，连接成功才能通信。服务端不可以主动连接客户端

* 在Java中，提供了两个类用于实现TCP通信程序

* 客户端： java.net.Socket 类表示。创建 Socket 对象，向服务端发出连接请求，服务端响应请求，两者建立连接开始通信

* 服务端： java.net.ServerSocket 类表示。创建 ServerSocket 对象，相当于开启一个服务，并等待客户端的连接。

* 能够掌握Socket和ServerSocket类

* Socket

* ServerSocket

* 能够掌握网络编程HelloWorld开发

```
案例一：服务端等待客户端连接，客户端向服务端传输数据
 * 服务端
public class Server {
    public static void main(String[] args) throws IOException {
        // 构建ServerSocket
        ServerSocket serverSocket=new ServerSocket(8888);
        System.out.println("服务端启动成功...");
        // 等待客户端连接
        Socket server = serverSocket.accept();
        System.out.println("处理请求IP为:"+server.getInetAddress().getHostAddres
        // 获得输入流
        InputStream is = server.getInputStream();
        // 处理输入流的内容
        byte[] buffer=new byte[1024];
        int len = is.read(buffer);
        String msg=new String(buffer,0,len);
        System.out.println("msg = " + msg);
        // 释放资源
        is.close();
        serverSocket.close();
    }
}
 * 客户端
public class Client {
    public static void main(String[] args) throws IOException {
        // 构建客户端Socket
```

```
        Socket client=new Socket("127.0.0.1",8888);
        System.out.println("客户端启动成功");
        // 获得输出流
        OutputStream os = client.getOutputStream();
        // 向服务端输出内容
        os.write("亮哥教育，做教育我们是认真的".getBytes());
        // 释放资源
        os.close();
        client.close();
    }
}
 * 测试：先启动服务端等待客户端的链接
 结果：
 服务端：
 服务端启动成功...
 处理请求IP为:127.0.0.1
 msg = 亮哥教育，做教育我们是认真的
 客户端：
 客户端启动成功

 * 案例二：客户端和服务端互相传递数据
  *服务端
public class Server {
    public static void main(String[] args) throws IOException {
        // 构建ServerSocket
        ServerSocket serverSocket=new ServerSocket(8888);
        System.out.println("服务端启动成功...");
        // 等待客户端连接
        Socket server = serverSocket.accept();
        System.out.println("处理请求IP为:"+server.getInetAddress().getHostAddres
        // 获得输入流
        InputStream is = server.getInputStream();
        // 处理输入流的内容
        byte[] buffer=new byte[1024];
        int len = is.read(buffer);
        String msg=new String(buffer,0,len);
        System.out.println("msg = " + msg);

        //想客户端传输数据
        OutputStream os = server.getOutputStream();
```

```
        os.write("可以".getBytes());
        // 释放资源
        is.close();
        serverSocket.close();
    }
}
 * 客户端
public class Client {
    public static void main(String[] args) throws IOException {
        // 构建客户端Socket
        Socket client=new Socket("127.0.0.1",8888);
        System.out.println("客户端启动成功");
        // 获得输出流
        OutputStream os = client.getOutputStream();
        // 向服务端输出内容
        os.write("请问我可以访问您嘛?".getBytes());

        InputStream is = client.getInputStream();
        byte[] buffer=new byte[1024];
        int len = is.read(buffer);
        String result=new String(buffer,0,len);
        System.out.println("result = " + result);
        // 释放资源
        os.close();
        client.close();
    }
}
 * 测试效果

```

* 能够掌握网络编程案例之文件上传

```
* 服务端
public class Server {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket=new ServerSocket(8888);
        System.out.println("服务器启动成功...");
        while(true){
            Socket server = serverSocket.accept();
```

```
            System.out.println("客户端请求接入...");
            ThreadPoolUtils.getPool().execute(new Runnable() {
                @Override
                public void run() {
                    try {
                        BufferedInputStream bis=new BufferedInputStream(server.
                        BufferedOutputStream bos=new BufferedOutputStream(new F
                        byte[] buffer=new byte[1024];
                        int len=-1;
                        while((len=bis.read(buffer))!=-1){
                            bos.write(buffer,0,len);
                        }
                        // 告诉客户端文件上传成功
                        OutputStream os = server.getOutputStream();
                        os.write("文件上传成功".getBytes());
                        System.out.println("文件保存成功");
                        bos.close();
                        bis.close();
                        server.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
    }
}

* 客户端
public class Client {
    public static void main(String[] args)  {
        ThreadPoolUtils.getPool().execute(new Runnable() {
            @Override
            public void run() {
                // 构建客户端Socket
                try {
                    Socket client = new Socket("127.0.0.1",8888);
                    System.out.println("客户端启动成功");
                    // 获得输出流
                    BufferedOutputStream bos=new BufferedOutputStream(client.ge
```

```
                    BufferedInputStream bis=new BufferedInputStream(new FileInp
                    byte[] buffer=new byte[1024];
                    int len=-1;
                    while((len=bis.read(buffer))!=-1){
                        bos.write(buffer,0,len);
                    }
                    bos.flush();
//                    // 关闭输出流,通知服务端,写出数据完毕
                    client.shutdownOutput();
                    // 获得服务端传输的数据
                    InputStream is = client.getInputStream();
                    byte[] b=new byte[1024];
                    len = is.read(b);
                    String result=new String(b,0,len);
                    System.out.println("result = " + result);
                    bis.close();
                    client.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
        ThreadPoolUtils.getPool().shutdown();
    }
}
```

* 能够掌握网络编程案例之编写简易版tomcat

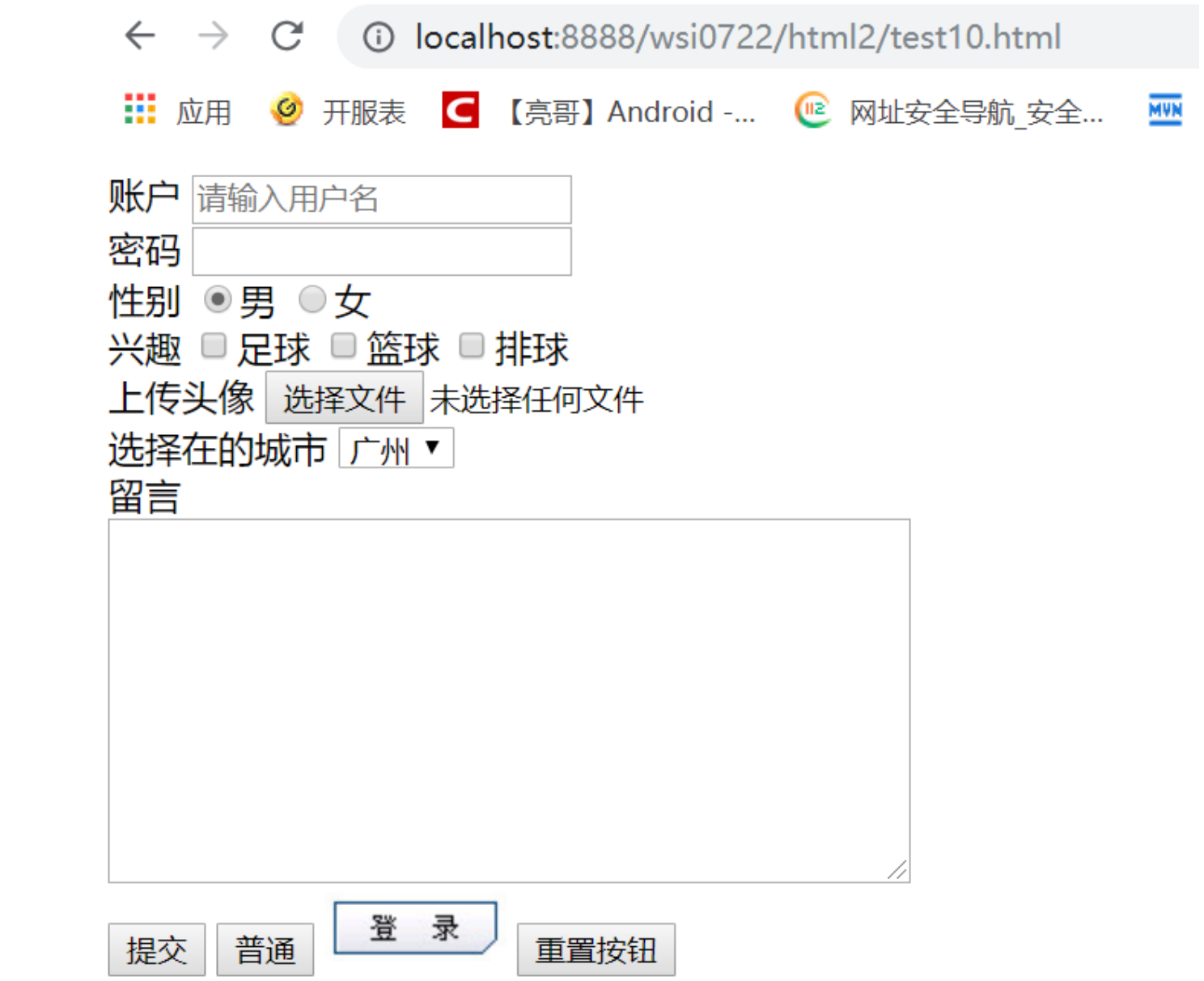
```
案例一：获取浏览器访问服务器的请求信息
 服务器端
public class WebServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket=new ServerSocket(8888);
        System.out.println("tomcat 服务器启动成功...");
        while(true){
            Socket server = serverSocket.accept();
            ThreadPoolUtils.getPool().execute(new Runnable() {
```

```
                @Override
                public void run() {
                    try {
                        InputStream is = server.getInputStream();
                        byte[] b=new byte[1024];
                        int len = is.read(b);
                        String msg=new String(b,0,len);
                        System.out.println(msg);
                        is.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
    }
}

 * 复制之前html2的项，放在项目下
 * 访问：http://localhost:8888/wsi0722/html2/test1.html
 * 结果：
tomcat 服务器启动成功...
GET /wsi0722/html2/test1.html HTTP/1.1
Host: localhost:8888
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
Sec-Fetch-User: ?1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en-AU;q=0.8,en;q=0.7
Cookie: Hm_lvt_aa5c701f4f646931bf78b6f40b234ef5=1554828318,1554945605; Webstorm

* 案例二：服务器
public class WebServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket=new ServerSocket(8888);
        System.out.println("tomcat 服务器启动成功...");
```

```
        while(true){
            Socket server = serverSocket.accept();
            ThreadPoolUtils.getPool().execute(new Runnable() {
                @Override
                public void run() {
                    try {
                        BufferedReader br=new BufferedReader(new InputStreamRea
                        String request = br.readLine();
                        String path=request.split(" ")[1];
                        System.out.println(path);
                        FileInputStream is=new FileInputStream(path);
                        OutputStream os = server.getOutputStream();
                        // 写入HTTP协议响应头,固定写法
                        os.write("HTTP/1.1 200 OK\r\n".getBytes());
                        os.write("Content-Type:text/html\r\n".getBytes());
                        // 必须要写入空行,否则浏览器不解析
                        os.write("\r\n".getBytes());
                        byte[] buffer=new byte[1024];
                        int len=-1;
                        while((len=is.read(buffer))!=-1){
                            os.write(buffer,0,len);
                        }
                        os.flush();
                        // 释放资源
                     is.close();
                     os.close();
                     br.close();
                     server.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
    }
}
* 复制之前html2的项，放在项目下
* 访问：http://localhost:8888/wsi0722/html2/test10.html
* 结果可以下浏览器上看到效果
```


localhost:8888/wsi0722/html2/test10.html
应用
开服表
【亮哥】Android -...
网址安全导航_安全...
账户
请输入用户名
密码
性别
男
女
兴趣
足球
篮球
排球
上传头像
选择文件
未选择任何文件
选择在的城市
广州
留言
提交
普通
登 录
重置按钮