2021计网第二十五组大作业报告

成员分工

姓名	学号	分工
陈梓俊	191250016	POST请求，登录注册功能，状态码
顾龙	191250037	Server框架，GET请求，状态码
冯国豪	191250031	Client全部内容（视频录制，文档）
刘庭烽	191250093	长连接实现，状态码，Server视频录制
丁云翔	191250026	Server文档、状态码实现、长连接实现

项目地址

https://github.com/strategic-zjc/socket-programming/

项目简介

基于 Java Socket API 搭建简单的 HTTP 客户端和服务器端程序，实现如下：

1. 实现基础的 HTTP 请求、响应功能，具体要求如下：
    - HTTP 客户端可以发送请求报文、呈现响应报文（命令行和 GUI 都可以）
    - HTTP 客户端对 301、302、304 的状态码做相应的处理
    - HTTP 服务器端支持 GET 和 POST 请求
    - HTTP 服务器端支持 200、301、302、304、404、405、500 的状态码
    - HTTP 服务器端实现长连接
    - MIME 至少支持三种类型，包含一种非文本类型
2. 基于以上的要求，实现注册，登录功能(数据无需持久化，存在内存中即可， 只需要实现注册和登录的接口，可以使用 postman 等方法模拟请求发送，无需客 户端)。

HttpServer

我们实现了基础的HTTP请求、响应功能，服务器端支持GET和POST请求、支持200、301、302、304、404、405、500的状态码、支持长连接，MIME支持三种类型，并且实现了注册和登录功能。

服务器端代码主体部分

服务器类位于SimpleServer.java中，负责创建一个服务器对象并运行，服务器对象创建socket和线程来处理客户端的请求，代码实现如下：

```
public class SimpleServer {
    public static ArrayList<BasicExecutor> Executors = new ArrayList<>();
    public static Timer timer = new Timer();
    public static void main(String[] args) {
        SimpleServer server = new SimpleServer();
        server.go();
    }

    private void go(){

        Executors.add(new LoginExecutor());
        Executors.add(new RegisterExecutor());
        Executors.add(new ErrorExecutor());
       // Executors.add(new StaticResourceHandler());
        try {
            ServerSocket serverSocket = new ServerSocket(5000);// 先创建一个
            System.out.println("http://localhost:5000");

            while(true){
                Socket socket = serverSocket.accept();

                Thread readThread = new Thread(new ClientHandler(socket));

                readThread.start();
                System.out.println("Got a connection from " +
socket.getInetAddress().getHostAddress());
            }
        }catch (Exception e){
            e.printStackTrace();
        }

    }
}
```

服务器在启动时会创建一个ClientHandler线程，来处理客户端请求，客户端传来的请求的处理主要在ClientHandler.java中，具体代码实现如下：

```
public class ClientHandler implements Runnable {
    // new thread
    BufferedReader inFromClient;
    DataOutputStream outToClient;
    Socket socket;
    boolean ServerSwitch;
    boolean isTimeout = false;
    public static TimerTask timerTask = null;


    public ClientHandler(Socket clientSock) {
        try {
            socket = clientSock;
            inFromClient = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            outToClient = new DataOutputStream(socket.getOutputStream());
            ServerSwitch = true;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    /**
     * web server
     */
    @Override
    public void run() {
        try {

            while (true) {
                if (isTimeout) {
                    socket.close();
                    return;
                }
                // read all bytes from socket stream
                String line;
                StringBuilder sb = new StringBuilder();
                while ((line = inFromClient.readLine()) != null) {
                    sb.append(line).append('\n');
                    if (line.isEmpty())
                        break;
                }
                if (sb.toString().equals("")) return;

                System.out.println(sb.toString());

                HttpRequest request = Util.String2Request(sb.toString());

                if (request.getHeaders().getValue("Keep-Alive") != null) {
                    String timeout = request.getHeaders().getValue("Keep-
Alive");
                    if (timerTask != null) {
                        timerTask.cancel();
```

```
                    }
                    timerTask = new TimerTask() {
                        @Override
                        public void run() {
                            isTimeout = true;
                        }
                    };
                    SimpleServer.timer.schedule(timerTask,
Integer.parseInt(timeout.substring(8)) * 1000L);
                }


                String contentLength = request.getHeaders().getValue("Content-
Length");

                if (contentLength != null) {

                    int length = Integer.parseInt(contentLength);

                    char[] cbuf = new char[length];
                    inFromClient.read(cbuf, 0, length);

                    request.getBody().setData(String.valueOf(cbuf));
                }


                String target = request.getStartLine().getTarget();
                String method = request.getStartLine().getMethod();

                HttpResponse response = null;
                BasicExecutor executor = null;

                // 如果请求一个静态资源，调用StaticResourceHandler
                if (StaticResourceHandler.isStaticTarget(target) &&
method.toLowerCase().equals("get")) {
                    executor = new StaticResourceHandler();
                } else {
                    // 否则，在持有的executor中找到合适的，用这个executor处理请求
                    for (BasicExecutor e : SimpleServer.Executors) {
                        if (target.endsWith(e.getUrl()) &&
method.toLowerCase().equals(e.getMethod().toLowerCase())) {
                            executor = e;
                            break;
                        }
                    }
                }

                // 找不到合适的executor
                // 404: 没有对应的url 405: 有对应的url但是没有对应的method
                if (executor == null) {
                    response = Template.generateStatusCode_404();
                    for (BasicExecutor e : SimpleServer.Executors) {
                        if (target.endsWith(e.getUrl())) {
                            response = Template.generateStatusCode_405();
                            break;
                        }
                    }
```

```
                } else {
                    response = executor.handle(request);
                }


                outToClient.write(response.ToBytes());
                //timer 如果再次收到请求，重置timer，否则就关闭

            }
//              outToClient.close();
        }catch (SocketException e){

        } catch (Exception e) {
            HttpResponse response = Template.generateStatusCode_500();
            try {
                outToClient.write(response.ToBytes());
            }catch (Exception ee){}
            e.printStackTrace();
        }
    }
}
```

GET和POST请求实现

支持处理GET和POST请求，对不同的请求采取对应的处理方法，代码实现在ClientHandler.java中，具体代码实现如下：

```
String target = request.getStartLine().getTarget();
String method = request.getStartLine().getMethod();

HttpResponse response = null;
BasicExecutor executor = null;

// 如果请求一个静态资源，调用StaticResourceHandler
if (StaticResourceHandler.isStaticTarget(target) &&
method.toLowerCase().equals("get")) {
    executor = new StaticResourceHandler();
}
else{
    // 否则，在持有的executor中找到合适的，用这个executor处理请求
    for (BasicExecutor e : SimpleServer.Executors) {
        if (target.endsWith(e.getUrl()) &&
method.toLowerCase().equals(e.getMethod().toLowerCase())) {
            executor = e;
            break;
        }
    }
}
```

200、301、302、304、404、405、500的状态码实现

我们把所有状态码的处理放在一个模板类里，并在对应的条件下调用对应的状态码生成函数来处理，模板类实现如下：

```
public class Template {
```

```
    public static HttpResponse generateStatusCode_200(String hint){
        StatusLine statusLine = new StatusLine(1.1, StatusCode.OK.getCode(),"200
OK");
        Headers headers = new Headers();
        String html_200 = "<html>\n" +
                "<head><title>200 OK</title></head>\n" +
                "<body bgcolor=\"white\">\n" +
                "<center><h1>200 OK</h1><h2>" + hint +"</h2><h6>simple http-
server<h6></center>\n" +
                "</body>\n" +
                "</html>";
        headers.addHeader("Content-Type", "text/html");
        headers.addHeader("Content-Length", Long.toString(html_200.length()));
        Body body = new Body(html_200);
        return new HttpResponse(statusLine, headers, body);
    }
    public static HttpResponse generateStatusCode_404(){
        StatusLine statusLine = new StatusLine(1.1,
StatusCode.NOT_FOUND.getCode(), "404 Method Not Allowed");
        Headers headers = new Headers();
        String html404 = "<html>\n" +
                "<head><title>404 Not Found</title></head>\n" +
                "<body bgcolor=\"white\">\n" +
                "<center><h1>404 Not Found</h1><h6>simple http-server<h6>
</center>\n" +
                "</body>\n" +
                "</html>";
        headers.addHeader("Content-Type", "text/html");
        headers.addHeader("Content-Length", Long.toString(html404.length()));
        Body body = new Body(html404);
        return new HttpResponse(statusLine, headers, body);
    }

    public static HttpResponse generateStatusCode_405(){
        StatusLine statusLine = new StatusLine(1.1,
StatusCode.METHOD_NOT_ALLOWED.getCode(),"405 Method Not Allowed");
        Headers headers = new Headers();
        String html405 = "<html>\n" +
                "<head><title>405 Not Allowed</title></head>\n" +
                "<body bgcolor=\"white\">\n" +
                "<center><h1>405 Not Allowed</h1><h6>simple http-server<h6>
</center>\n" +
                "</body>\n" +
                "</html>";
        headers.addHeader("Content-Type", "text/html");
        headers.addHeader("Content-Length", Long.toString(html405.length()));
        Body body = new Body(html405);
        return new HttpResponse(statusLine, headers, body);
    }

    public static HttpResponse generateStatusCode_500(){
        StatusLine statusLine = new StatusLine(1.1,
StatusCode.INTERNAL_SERVER_ERROR.getCode(),"500 Internal Server Error");
        Headers headers = new Headers();
        String html_500 = "<html>\n" +
                "<head><title>500 Internal Server Error</title></head>\n" +
                "<body bgcolor=\"white\">\n" +
```

```
                "<center><h1>500 Internal Server Error</h1><h6>simple http-
server<h6></center>\n" +
                "</body>\n" +
                "</html>";
        headers.addHeader("Content-Type", "text/html");
        headers.addHeader("Content-Length", Long.toString(html_500.length()));
        Body body = new Body(html_500);
        return new HttpResponse(statusLine, headers, body);
    }
    public static HttpResponse generateStatusCode_400(){
        StatusLine statusLine = new StatusLine(1.1,
StatusCode.INTERNAL_SERVER_ERROR.getCode(),"400 Bad Request");
        Headers headers = new Headers();
        String html_400 = "<html>\n" +
                "<head><title>400 Bad Request</title></head>\n" +
                "<body bgcolor=\"white\">\n" +
                "<center><h1>400 Bad Request</h1><h6>simple http-server<h6>
</center>\n" +
                "</body>\n" +
                "</html>";
        headers.addHeader("Content-Type", "text/html");
        headers.addHeader("Content-Length", Long.toString(html_400.length()));
        Body body = new Body(html_400);
        return new HttpResponse(statusLine, headers, body);
    }

    public static HttpResponse generateStatusCode_304(){
        StatusLine statusLine = new StatusLine(1.1,
StatusCode.NOT_MODIFIED.getCode(),"304 Not Modified");
        Headers headers = new Headers();
        Body body = new Body();
        return new HttpResponse(statusLine, headers, body);
    }

    public static HttpResponse generateStatusCode_301(String url){
        StatusLine statusLine = new StatusLine(1.1,
StatusCode.MOVED_PERMANENTLY.getCode(), "301 Moved Permanrntly");
        Headers headers = new Headers();
        String html_301 = "<html>\n" +
                "<head><title>301 Moved Permanrntly</title></head>\n" +
                "<body bgcolor=\"white\">\n" +
                "<center><h1>301 Moved Permanrntly</h1><h6>simple http-server<h6>
</center>\n" +
                "</body>\n" +
                "</html>";
        headers.addHeader("Content-Type", "text/html");
        headers.addHeader("Content-Length", Long.toString(html_301.length()));
        headers.addHeader("Location", url);
        Body body = new Body(html_301);
        return new HttpResponse(statusLine, headers, body);
    }

    public static HttpResponse generateStatusCode_302(String url){
        StatusLine statusLine = new StatusLine(1.1, StatusCode.FOUND.getCode(),
"302 Found");
        Headers headers = new Headers();
        String html_302 = "<html>\n" +
                "<head><title>302 Found</title></head>\n" +
```

```
                "<body bgcolor=\"white\">\n" +
                "<center><h1>302 Found</h1><h6>simple http-server<h6></center>\n"
 +
                "</body>\n" +
                "</html>";
        headers.addHeader("Content-Type", "text/html");
        headers.addHeader("Content-Length", Long.toString(html_302.length()));
        headers.addHeader("Location", url);
        Body body = new Body(html_302);
        return new HttpResponse(statusLine, headers, body);
    }
}
```

200状态码

服务器成功处理请求时，返回200状态码，例如在登陆成功时：

```
if (db.containsKey(username) && db.get(username).equals(password)) {
    String hint = "You have successfully login in!";
    response = Template.generateStatusCode_200(hint);
}
```

301和302状态码

301和302状态码分别代表被请求的资源已永久或临时移动到新位置，例如在获取静态资源时，若资源已永久或临时移动到新位置，则返回对应的状态码：

```
if (MovedPermanentlyResource.containsKey(target)) {
    return
Template.generateStatusCode_301(MovedPermanentlyResource.get(target));
}
else if (MovedTemporarilyResource.containsKey(target)) {
    return
Template.generateStatusCode_302(MovedTemporarilyResource.get(target));
}
```

304状态码

304状态码代表未修改。例如上次请求后，请求的网页未修改过，服务器处理此请求时，不会再次返回请求的资源：

```
// add last modified
Date fileLastModifiedTime = new Date(f.lastModified());
SimpleDateFormat sdf = new SimpleDateFormat("EEE, dd MMM yyyy hh:mm:ss z",
Locale.ENGLISH);
sdf.setTimeZone(TimeZone.getTimeZone("GMT"));
System.out.println(sdf.format(fileLastModifiedTime));
headers.addHeader("Last-Modified", sdf.format(fileLastModifiedTime));

String time = request.getHeaders().getValue("If-Modified-Since");
if (time != null){
    Date Limit =  sdf.parse(time);
    if(Limit.compareTo(fileLastModifiedTime) > 0){
        return Template.generateStatusCode_304();
    }
}
```

404和405状态码

404状态码代表服务器没有对应的url，405状态码代表服务器有对应的url但是没有对应的method。例如：

```
// 找不到合适的executor
// 404: 没有对应的url 405: 有对应的url但是没有对应的method
if (executor == null) {
    response = Template.generateStatusCode_404();
        for (BasicExecutor e : SimpleServer.Executors) {
            if (target.endsWith(e.getUrl())) {
                response = Template.generateStatusCode_405();
                break;
            }
        }
}
```

500状态码

500状态码代表服务器遇到错误，无法完成请求。例如服务器出现异常：

```
catch (Exception e) {
    HttpResponse response = Template.generateStatusCode_500();
    try {
        outToClient.write(response.ToBytes());
        }catch (Exception ee){}
        e.printStackTrace();
}
```

长连接的实现

长连接主要依靠一个isTimeout变量和一个TimerTask对象来实现，建立连接后保持一段时间，如果这段时间内没有新的请求，则断开连接，代码实现位于ClientHandler.java中，相关代码如下

```
if (isTimeout) {
    socket.close();
    return;
}

...

if (request.getHeaders().getValue("Keep-Alive") != null) {
    String timeout = request.getHeaders().getValue("Keep-Alive");
    if (timerTask != null) {
        timerTask.cancel();
    }
    timerTask = new TimerTask() {
        @Override
        public void run() {
            isTimeout = true;
        }
    };
    SimpleServer.timer.schedule(timerTask,
Integer.parseInt(timeout.substring(8)) * 1000L);
}
```

MIME媒体类型的实现

对媒体类型的实现主要在StaticResourceHandler.java里，实现了.html, .png, .javascript三种类型的文件传输，代码如下：

```
public class StaticResourceHandler extends BasicExecutor {

    public static HashMap<String, String> MovedPermanentlyResource = new
HashMap<>();
    public static HashMap<String, String> MovedTemporarilyResource = new
HashMap<>();
    public static HashMap<String, String> ModifiedTime = new HashMap<>();

    public StaticResourceHandler() {
        MovedPermanentlyResource.put("/movedPic.png", "/pic.png");
        MovedPermanentlyResource.put("/movedIndex.html", "/index.html");
        MovedTemporarilyResource.put("/movedPic2.png", "/pic.png");
        MovedTemporarilyResource.put("/movedIndex2.html", "/index.html");
    }

    public static boolean isStaticTarget(String target) {
        target = target.substring(target.lastIndexOf("/") + 1);
        return target.contains(".");
    }

    public HttpResponse handle(HttpRequest request) throws Exception{
        StatusLine statusLine = null;
        Headers headers = new Headers();
        Body body = new Body();
        String target = request.getStartLine().getTarget();

        String host = headers.getValue("Host");

        if (MovedPermanentlyResource.containsKey(target)) {
            return
Template.generateStatusCode_301(MovedPermanentlyResource.get(target));
        }
        else if (MovedTemporarilyResource.containsKey(target)) {
            return
Template.generateStatusCode_302(MovedTemporarilyResource.get(target));
        }
        else {
            statusLine = new StatusLine(1.1, 200, "OK");
        }

        if (target.endsWith(".html")) {
            headers.addHeader("Content-Type", "text/html");
        } else if (target.endsWith(".png")) {
            headers.addHeader("Content-Type", "image/png");
        }else if(target.endsWith(".js")){
            headers.addHeader("Content-Type", " text/javascript");
        }

        String path = target.substring(target.lastIndexOf("/") + 1);
```

```
        // add length
        File f = new File(path);
        headers.addHeader("Content-Length", Long.toString(f.length()));

        // add last modified
        Date fileLastModifiedTime = new Date(f.lastModified());
        SimpleDateFormat sdf = new SimpleDateFormat("EEE, dd MMM yyyy hh:mm:ss
z", Locale.ENGLISH);
        sdf.setTimeZone(TimeZone.getTimeZone("GMT"));
        System.out.println(sdf.format(fileLastModifiedTime));
        headers.addHeader("Last-Modified", sdf.format(fileLastModifiedTime));

        String time = request.getHeaders().getValue("If-Modified-Since");
        if (time != null){
            Date Limit =  sdf.parse(time);
            if(Limit.compareTo(fileLastModifiedTime) > 0){
                return Template.generateStatusCode_304();
            }
        }

        byte[] bytesArray = new byte[(int) f.length()];
        try {
            FileInputStream fis = new FileInputStream(f);
            fis.read(bytesArray);
            //read file into bytes[]
            fis.close();
        } catch (Exception e) {
            e.printStackTrace();
            return new HttpResponse(new StatusLine(1.1,
StatusCode.NOT_FOUND.getCode(), "Not Found"), new Headers(), new Body());
        }


        body.setData(bytesArray);

        return new HttpResponse(statusLine, headers, body);
    }
}
```

注册和登录功能的实现

注册登录功能的实现主要在RegisterExecutor.java和LoginExecutor.java中，读取请求中携带的用户名和密码，检验请求是否符合语法，并根据核验结果返回对应的结果和状态码，代码实现如下：

```
public class RegisterExecutor extends BasicExecutor{

    public static HashMap<String, String> db = new HashMap<>();

    public RegisterExecutor(){
        this.url = "/register";
        this.method = "post";
    }

    @Override
    public HttpResponse handle(HttpRequest request) {
        HashMap<String, String> db = RegisterExecutor.db;
        HttpResponse response = null;
```

```
        Headers headers = request.getHeaders();
        String contentType = headers.getValue("Content-Type").split(";")
[0].trim();
        Body body = request.getBody();

        if(!contentType.equals("application/x-www-form-urlencoded")){
            response = Template.generateStatusCode_405();
            return response;
        }

        String[] key_val = body.ToString().split("&");
        assert (key_val.length == 2);
        String username = null;
        String password = null;
        for(int i = 0; i < key_val.length; i++){
            String[] tmp = key_val[i].split("=");
            assert (tmp.length == 2);
            if(tmp[0].equals("username")){
                username = tmp[1].trim();
            }else if (tmp[0].equals("password")){
                password = tmp[1].trim();
            }
        }
        if(username == null || password == null){
            response = Template.generateStatusCode_405();
        }else {
            if (!db.containsKey(username)) {
                db.put(username, password);
                String hint = "You have successfully register!";
                response = Template.generateStatusCode_200(hint);
            } else {
                String hint = "You have successfully register!";
                response = Template.generateStatusCode_200(hint);
            }
        }

        return response;
    }
}
```

```
public class LoginExecutor extends BasicExecutor{

    public LoginExecutor(){
        this.url = "/login";
        this.method = "post";
    }

    @Override
    public HttpResponse handle(HttpRequest request) {
        HashMap<String, String> db = RegisterExecutor.db;
        HttpResponse response = null;
        Headers headers = request.getHeaders();
        String contentType = headers.getValue("Content-Type").split(";")
[0].trim();
        Body body = request.getBody();

        if(!contentType.equals("application/x-www-form-urlencoded")){
```

```
            response = new HttpResponse(new StatusLine(1.1, 400, "Bad Request"),
new Headers(), new Body());
            return response;
        }

        String[] key_val = body.ToString().split("&");
        assert (key_val.length == 2);
        String username = null;
        String password = null;
        for(int i = 0; i < key_val.length; i++){
            String[] tmp = key_val[i].split("=");
            assert (tmp.length == 2);
            if(tmp[0].equals("username")){
                username = tmp[1].trim();
            }else if (tmp[0].equals("password")){
                password = tmp[1].trim();
            }
        }
        if(username == null || password == null){
            response = new HttpResponse(new StatusLine(1.1, 400, "Bad Request"),
new Headers(), new Body());
        }else {

            if (db.containsKey(username) && db.get(username).equals(password)) {
                String hint = "You have successfully login in!";
                response = Template.generateStatusCode_200(hint);
            } else {
                String hint = "login failed";
                response = Template.generateStatusCode_200(hint);
            }
        }

        return response;
    }
}
```

HttpClient

功能列表

1. 基本的客户端收发功能（使用API接口调用和使用CommandLine两种形式展示）
2. 客户端keep-alive支持
3. 301，302，304状态码支持
4. 文本类型文件接收与显示支持
5. 任意类型body保存为文件支持，从文件中读入body支持
6. 特殊格式body的解析与保存（以下载百度站点为例）
7. 实现了不同级别的日志保存与显示，便于观察客户端行为
8. 客户端存储隔离，可以在使用api调用时启动多个客户端，从而隔离客户端行为

功能展示

基本客户端收发功能

```
GET / HTTP/1.1
Host: ditu.yjdy.org

```

上面是一个基本的http Get请求，我们可以使用命令行模式直接输入上面的内容，按下回车便可以发送请求，并接收完整的响应文本

同样，对于一个基本的http Post 请求，也可以这样使用

下面是服务端实现的注册登录功能

```
POST /login HTTP/1.1
Host: localhost:5000
Content-Type: application/x-www-form-urlencoded
Content-Length: 30

username=admin&password=123456
```

控制台客户端如果不手动停止会一直接收输入

在输入请求后，客户端会对请求的合法性做出一定检查，该选项针对api调用和控制台输入都生效，这是因为使用控制台标准输入输出和使用API调用复用了同一套代码，所以他们的行为是一致的。

对于api请求，一般的格式是这样的

```
RequsetLine requsetLine = new RequsetLine(Method.GET,"/");//创建请求方法行，指明请求的地址和方法
MessageHeader messageHeader = new MessageHeader();//创建请求头，并向请求头中添加内容
messageHeader.put(Header.Host,"www.baidu.com");
messageHeader.put(Header.Accept,"*/*");
messageHeader.put(Header.Connection,"keep-alive");
messageHeader.put(Header.Accept_Encoding,"gzip, deflate, br");
MessageBody messageBody= new MessageBody();//创建请求体，向请求体中添加信息
HttpRequest httpRequest = new
HttpRequest(requsetLine,messageHeader,messageBody);
Client client = new Client();
try {
    HttpResponse httpResponse =client.sendHttpRequest(httpRequest);//发送请求，接收response
    httpResponse.saveBody("./123.html");//将收到的数据保存在本地
} catch (Exception e) {
    e.printStackTrace();//处理异常
}
```

客户端基于`maven`构建，可以方便地被集成到其他项目之中，也可以直接在控制台使用

客户端keep-alive支持

下面的内容将使用java api调用测试来展示

我们连续调用6次同一请求，其中前三次不携带keep-alive头，后三次携带keep-alive头,发现前三次请求时都会创建新的连接，第四次也会创建，第五次和第六次则不会创建。

301,302,304

301

服务端地址`localhost:5000/movedPic.png`已经永久变更为了`localhost:5000/pic.png`,我们尝试进行两次请求

发现第二次请求时不会请求`movedPic.png`,而是直接请求`pic.png`

302

服务端地址`localhost:5000/movedIndex2.html`已经临时变更为了`localhost:5000/index.html`,我们尝试进行两次请求

发现第二次请求时仍然会请求`movedIndex2.html`,接收到302码进行重定向

304

```
GET /images/zl_bg5.png HTTP/1.1
Host: www.historychina.net
```

这个网站实现了304重定向，可以对客户端行为进行观察。

特殊格式

使用Get请求访问baidu.com时，当携带`Accept-Encoding: gzip, deflate, br`时，百度默认会使用gzip的压缩方式以chunk的格式进行传输，client实现了对这一特殊格式的解析。

经过多次测试，在传输中遇到网络缓慢也能正常完成传输。

日志

客户端实现了DETAIL（显示包括完整报文在内的所有信息），INFO（显示一般日志信息），WARNING（显示警告信息），ERROR（显示错误信息）四个级别，便于使用与故障检查。