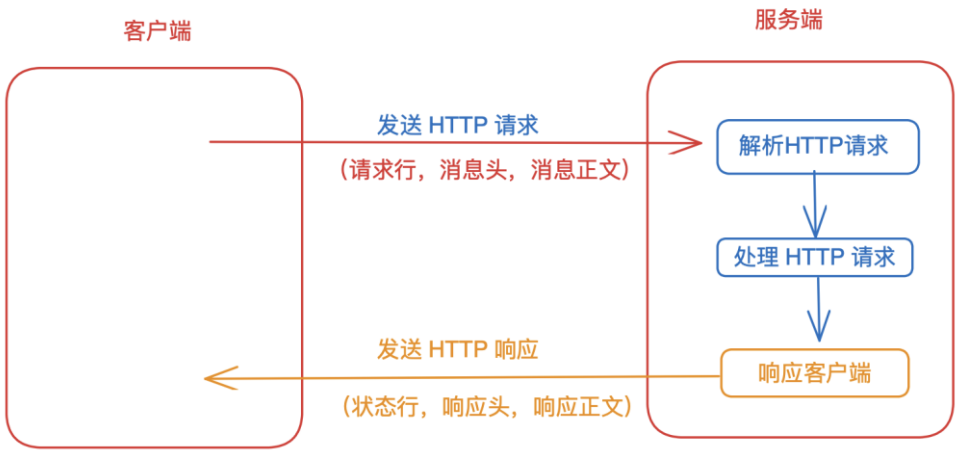
JavaSE-DAY11/DAY12 晚课

1. 客户端和浏览器交互



1.1 HTTP 请求(Request)

请求是浏览器发送给服务端的内容。一个请求由三部分组成:请求行，消息头，消息正文

请求行:

1. 请求行是一行字符串，包括三部分:
   a. 请求方法(GET/POST),
   b. 资源的路径,(index.html)
   c. 协议版本

例如:

GET /index.html HTTP/1.1(CRLF)

1. HTTP 协议规定一行字符串的结束是以(CRLF)结尾
   a. CR:回车符 ASC 编码对应值:13　回到本行的首端
   b. LF:换行符 ASC 编码对应值:10　换到下一行的相同位置

消息头:

1. 消息头由若干行组成，是客户端发送给服务端的一些附加信息。有的是用来告知服务端当前客户端自身的信息(比如使用什么浏览器，我的操作系统是什么等)，有的是用来维持通讯的一些处理操作信息等等。

2. 每一个消息头由一行组成，格式:name: value(CRLF)

注:消息头结束是以单独的 CRLF 表示，即:最后一个消息头发送完毕后，客户端会单独再发送一个 CRLF 表示消息头部分结束

例如

Host: localhost:8088(CRLF) key:value

Connection: keep-alive(CRLF)

Upgrade-Insecure-Requests: 1(CRLF)

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.162 Safari/537.36(CRLF)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,/*;q=0.8(CRLF)*

Accept-Encoding: gzip, deflate, br(CRLF)

Accept-Language: zh-CN,zh;q=0.9(CRLF)(CRLF)

最后一个消息头末尾时加一个 CRLF,表示整个消息头结

消息正文:

一个请求中可以不含有消息正文。消息正文是 2 进制数据，是用户提交给服务端的数据。它可能是一个表单数据(用户在页面输入的注册信息或者上传的图片等)，也可能是附件等。

1.2 HTTP 响应(Response)

响应是服务端发送给浏览器的内容，HTTP 协议对响应也有格式上的定义。

一个响应应当包含的内容有三部分:状态行,响应头,响应正文

状态行：

状态行也是一行字符串(以 CRLF 结尾)，包含三部分信息:

```
Java
protocol status_code status_reason(CRLF)
协议版本    状态代码      状态描述(CRLF)
```

其中状态代码是由一个三位数组成，分为 5 类:

1xx:保留

2xx:成功,指服务端成功处理请求

3xx:重定向,指服务端要求客户端再次发起请求到

指定资源

4xx:客户端错误

5xx:服务端错误

常见:

200:服务端成功处理并予以响应

404:客户端请求错误

500:服务端在处理请求时发生了错误

响应头:

响应头的格式与请求中的消息头一致，一行为一个响应头信息，并且在所有响应头发送完毕后要单独发送一个 CRLF 表示响应头部分结束。响应头是服务端发送给客户端的一些附加信息。

响应正文:

响应正文也是 2 进制数据，是服务端发送给客户端的数据，通常是客户端实际请求的资源(页面，图片等)。

请求是否含有消息正文和响应是否包含响应正文的标志都是看头信息中是否包含两个头:Content-Type 和 Content-Length

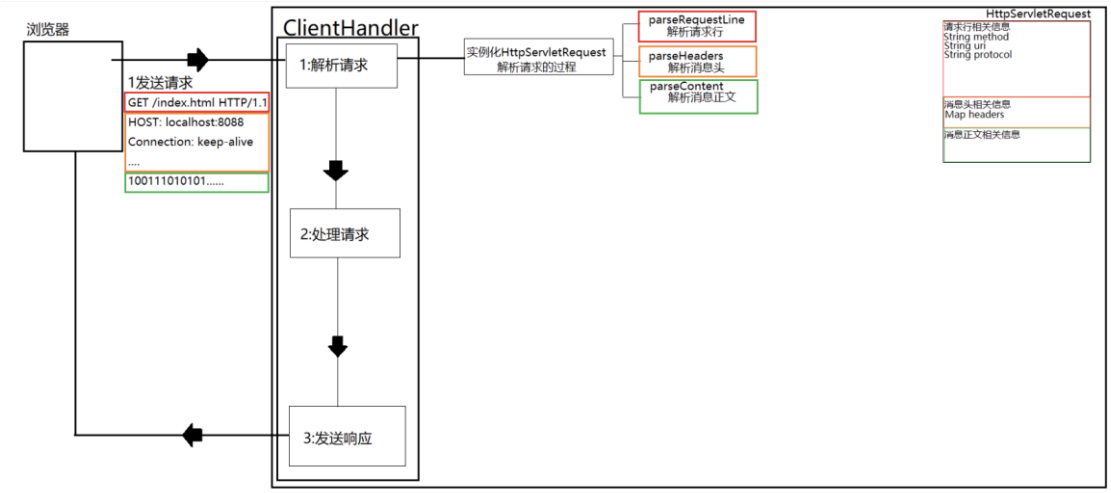
Content-Type:是用来说明正文的数据类型

Content-Length:是用来说明正文的数据长度共多少字节

2. 解析 HTTP 请求

- 主启动类： BirdBootApplication
  - 这个程序的功能是启动之后,可以使用浏览器来发送请求
- ClientHandler：这个类的作用是当有客户端连接到 WebServer 类后进行的一系列操作
  a. 解析请求
  b. 处理请求
  c. 发送响应

- HttpServletRequest：该类的每一个实例表示 HTTP 协议要求的浏览器发送给服务端的一个请求
    - *每个请求由三个部分构成:请求行、消息头、消息正文*



2.1 主启动类

```
Java
package core;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class BirdBootApplication {
    private ServerSocket serverSocket;

    public BirdBootApplication(){
        try {
            System.out.println("正在启动服务端...");
            serverSocket = new ServerSocket(8088);
            System.out.println("服务端启动完毕!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void start(){
        try {
```

```
            System.out.println("等待客户端连接...");
            Socket socket = serverSocket.accept();
            System.out.println("一个客户端连接了");
            //启动线程来处理该客户端交互
            ClientHandler handler = new ClientHandler(socket);
            //1:这里要传参 2:如果编译报错说明 ClientHandler 没有实现
Runnable 接口
            Thread t = new Thread(handler);
            t.start();


        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        BirdBootApplication application = new
BirdBootApplication();
        application.start();
    }
}
```

2.2 ClientHandler 类

```
Java
package core;

import http.HttpServletRequest;

import java.io.IOException;
import java.net.Socket;

/**
 * 该线程任务负责与指定的客户端(浏览器)完成一次 HTTP 交互
 * HTTP 协议要求浏览器与服务端采取一问一答的模式，因此这里的交互流程分为
三步：
 * 1：解析请求
```

```
 * 2：处理请求
 * 3：发送响应
 */
public class ClientHandler implements Runnable{
    private Socket socket;

    public ClientHandler(Socket socket){
        this.socket = socket;
    }

    public void run() {
        try {
            //1 解析请求
            HttpServletRequest request = new
HttpServletRequest(socket);

            String path = request.getUri();
            System.out.println("请求路径:"+path);

            //2 处理请求


            //3 发送响应


        } catch (IOException e) {
            e.printStackTrace();
        }


    }


}
```

2.3 HttpServletRequest 类

```
Java
package http;
```

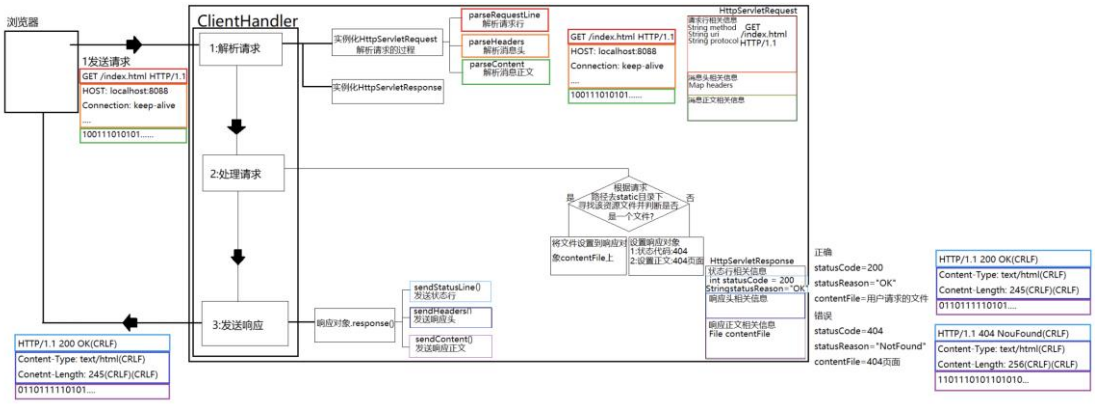
```
import java.io.IOException;
import java.io.InputStream;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;

/**
 * 请求对象
 * 该类的每一个实例表示 HTTP 协议要求的浏览器发送给服务端的一个请求
 * 每个请求由三个部分构成:
 * 请求行，消息头，消息正文
 *
 */
public class HttpServletRequest {
    //属性：用于存储和请求相关的数据
    //方法： 解析请求行，消息头，消息正文
    private Socket socket;
    //请求行相关信息
    private String method;//请求方式
    private String uri;//抽象路径
    private String protocol;//协议版本

    //消息头相关信息  key：value（Redis）
    private Map<String,String> headers = new HashMap<>();


    public HttpServletRequest(Socket socket) throws IOException {
        this.socket = socket;
        //1 解析请求行
        parseRequestLine();
        //2 解析消息头
        parseHeaders();
        //3 解析消息正文
        parseContent();
    }
    //解析请求行
    private void parseRequestLine() throws IOException {
```

```
        String line = readLine();
        System.out.println("请求行:"+line);

        String[] data = line.split("\\s");
        method = data[0];
        uri = data[1];
        protocol = data[2];

        System.out.println("method:"+method);//method:GET
        System.out.println("uri:"+uri);//uri:/index.html

System.out.println("protocol:"+protocol);//protocol:HTTP/1.1
    }
    //解析消息头
    private void parseHeaders() throws IOException {
        while(true) {//while 循环是因为浏览器发送多少个消息头不确定
            String line = readLine();
            if(line.isEmpty()){//如果 readLine 返回空字符串，说明单独
读取了回车+换行
                break;//因为单独的回车+换行表示消息头部分发送完毕
            }
            System.out.println("消息头:" + line);
            //将消息头按照冒号空格拆分为消息头名字和值并以 key,value 形
式保存到 headers 中
            String[] data = line.split(":\\s");
            headers.put(data[0],data[1]);
        }
        System.out.println("headers:"+headers);
    }
    //解析消息正文
    private void parseContent(){

    }



    /**
     * 读取客户端发送过来的一行字符串
```

```
     * @return
     */
    private String readLine() throws IOException {
        InputStream in = socket.getInputStream();
        char pre='a',cur='a';//pre 表示上次读取的字符，cur 表示本次读取的字符
        StringBuilder builder = new StringBuilder();//记录已读取的一行字符串的内容
        int d;//每次读取到的字节
        while((d = in.read()) != -1){
            cur = (char)d;//本次读取的字符
            if(pre==13 && cur==10){//是否已经连续读取到了回车+换行
                break;
            }
            builder.append(cur);//将本次读取的字符拼接
            pre = cur;//再下次读取前，将本次读取的字符记为上次读取的字符
        }
        //trim 的目的是将最后读取到的回车符去除
        return builder.toString().trim();
    }

    public String getMethod() {
        return method;
    }

    public String getUri() {
        return uri;
    }

    public String getProtocol() {
        return protocol;
    }

    /**
     * 根据消息头的名字获取对应的值
     * @param name
     * @return
     */
```

```
    public String getHeader(String name) {
        return headers.get(name);
    }
}
```

3. 解析 HTTP 响应



3.1 HttpServletResponse 类

```
Java
package http;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

/**
 * 响应对象
 * 该类的每一个实例用于表示 HTTP 协议规定的响应
 * 一个响应由三部分构成
 * 状态行，响应头，响应正文
 */
public class HttpServletResponse {
    private Socket socket;

    //状态行相关信息
```

```
    private int statusCode;//状态代码
    private String statusReason;//状态描述

    //响应头相关信息

    //响应正文相关信息
    private File contentFile;//正文对应的实体文件


    public HttpServletResponse(Socket socket){
        this.socket = socket;
    }

    /**
     * 响应方法
     * 该方法会将当前响应对象中的内容按照标准的 HTTP 响应格式发送给客户
端
     */
    public void response() throws IOException {
        //1 发送状态行
        sendStatusLine();
        //2 发送响应头
        sendHeaders();
        //3 发送响应正文
        sendContent();
    }
    //发送状态行
    private void sendStatusLine() throws IOException {
        println("HTTP/1.1"+" "+statusCode+" "+statusReason);
    }
    //发送响应头
    private void sendHeaders() throws IOException {
        println("Content-Type: text/html");
        println("Content-Length: " + contentFile.length());
        //单独发送回车+换行表达响应头部分发送完毕了
        println("");
    }
```

```
    //发送响应正文
    private void sendContent() throws IOException {
        OutputStream out = socket.getOutputStream();
        FileInputStream fis = new FileInputStream(contentFile);
        int len = 0;
        byte[] buf = new byte[1024*10];
        while((len = fis.read(buf))!=-1){
            out.write(buf,0,len);
        }
    }



    /**
     * 向客户端发送一行字符串
     * @param line
     */
    private void println(String line) throws IOException {
        OutputStream out = socket.getOutputStream();
        byte[] data = line.getBytes(StandardCharsets.ISO_8859_1);
        out.write(data);
        out.write(13);
        out.write(10);
    }

    public int getStatusCode() {
        return statusCode;
    }

    public void setStatusCode(int statusCode) {
        this.statusCode = statusCode;
    }

    public String getStatusReason() {
        return statusReason;
    }

    public void setStatusReason(String statusReason) {
        this.statusReason = statusReason;
    }

    public File getContentFile() {
        return contentFile;
```

```
    }

    public void setContentFile(File contentFile) {
        this.contentFile = contentFile;
    }
}
```

3.2 ClientHandler 类

```
Java
package core;

import http.HttpServletRequest;
import http.HttpServletResponse;

import java.io.File;
import java.io.IOException;
import java.net.Socket;
import java.net.URISyntaxException;

/**
 * 该线程任务负责与指定的客户端(浏览器)完成一次 HTTP 交互
 * HTTP 协议要求浏览器与服务端采取一问一答的模式，因此这里的交互流程分为
三步：
 * 1：解析请求
 * 2：处理请求
 * 3：发送响应
 */
public class ClientHandler implements Runnable{
    private Socket socket;

    public ClientHandler(Socket socket){
        this.socket = socket;
    }

    public void run() {
        try {
            //1 解析请求
            HttpServletRequest request = new
HttpServletRequest(socket);
```

```
            HttpServletResponse response = new
HttpServletResponse(socket);

            //2 处理请求
            String path = request.getUri();
            System.out.println("请求路径:"+path);
            File baseDir = new File(

ClientHandler.class.getClassLoader().getResource(".").toURI()
            );
            File staticDir = new File(baseDir,"static");
            File file = new File(staticDir,path);

            if(file.isFile()){
                response.setStatusCode(200);
                response.setStatusReason("OK");
                response.setContentFile(file);
            }else{
                response.setStatusCode(404);
                response.setStatusReason("NotFound");
                response.setContentFile(new
File(staticDir,"404.html"));
            }

            //3 发送响应
            response.response();

        } catch (IOException | URISyntaxException e) {
            e.printStackTrace();
        } finally {
            try {
                //按照 HTTP1.0 协议规则，一问一答后断开链接
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }
}
```