HTTP 协议

介绍

HTTP（hypertext transport protocol）协议也叫==超文本传输协议==，是一种基于 TCP/IP 的**应用层**通信协议，这个协议详细规定了浏览器和万维网服务器之间互相通信的规则。

协议主要规定内容：

- 客户端向服务器发送数据，称之为==请求报文==
- 服务器向客户端返回数据，称之为==响应报文==

Fiddle 工具

Fiddler 是一个http协议调试代理工具，使用它我们可以抓取网页的所有请求与响应，也就是咱们俗称的抓包。
进入软件之后-> tools -> options -> https -> 勾选左侧两个选项框 -> 弹出的窗口中 点击 『是』

请求

HTTP 请求报文组成

- 请求行
- 请求头
- 空行
- 请求体

请求行

有三个组成部分

- 请求类型： GET、POST、 DELETE
- 请求的URL： http://localhost:3000/index.html?username=sunwukong&password=123123
- 协议版本： HTTP/1.1

请求头

格式都是一致的 头名：头值

- Accept:设置客户端接受的数据类型
- Accept-Language:设置客户端接受的语言
- User-Agent:设置客户端的字符串的标识； 可以让服务器判断请求是哪种客户端发送的
- Accept-Encoding:设置客户端接受的压缩方式
- Host： 主机名
- Connection: 连接状态 keep-Alive 保持连接 close 关闭连接
- Cookie: 每次向服务器发送请求时，都会自动携带cookie发送请求

> 请求头的类型有很多，发现没有见过的请求头，可以查阅MDN

请求体

请求体的格式是非常灵活的，常见两种如下：

1. URL查询字符串形式（表单提交时请求体的类型就是这个类型）
2. JSON格式（AJAX请求时这种类型使用较多）

```
GET http://localhost:3000/index.html?username=sunwukong&password=123123 HTTP/1.1
Host: localhost:3000
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/64.0.3282.140 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
```

- GET http://localhost:3000/hello.html HTTP/1.1：GET请求，请求服务器路径为http://localhost:3000/hello.html，?后面跟着的是请求参数（查询字符串），协议是HTTP 1.1版本
- Host: localhost:3000：请求的主机名为localhost，端口号3000
- Connection: keep-alive：处理完这次请求后继续保持连接，默认为3000ms
- Pragma: no-cache：不缓存该资源，http 1.0的规定
- Cache-Control: no-cache： 不缓存该资源 http 1.1的规定，优先级更高
- Upgrade-Insecure-Requests: 1：告诉服务器，支持发请求的时候不用 http 而用 https
- User-Agent: Mozilla/5.0 (...：与浏览器和OS相关的信息。有些网站会显示用户的系统版本和浏览器版本信息，这都是通过获取User-Agent头信息而来的
- Accept: text/html,...：告诉服务器，当前客户端可以接收的文档类型。q 相当于描述了客户端对于某种媒体类型的喜好系数，该值的范围是 0-1。默认为1
- Accept-Encoding: gzip, deflate, br：支持的压缩格式。数据在网络上传递时，服务器会把数据压缩后再发送
- Accept-Language: zh-CN,zh;q=0.9：当前客户端支持的语言，可以在浏览器的工具选项中找到语言相关信息

URL的组成

- https://www.jokerps.com/?p=5812
  - 协议类型：　(例如：https 、http mongodb)
  - 域名：　www.jokerps.com
  - 端口号：　默认端口号443，不在URL中显示
  - 路径部分：　/（/开始（包括），？结尾（不包括））
  - 查询字符串：　p=5812

响应

HTTP响应报文组成

- 响应行
- 响应头
- 空行
- 响应体

响应行

- HTTP/1.1 200 OK
    - 协议版本 http/1.1
    - 响应状态码 404 找不到，500内部错误
        - 1xx 信息响应/临时响应
        - 2xx 成功响应
        - 3xx 重定向
        - 4xx 客户端错误
        - 5xx 服务器错误
    - 响应字符串 默认和状态码一一对应

响应头

格式是统一的 头的名字：头的值
响应头是可以自定义的
- Cache-Control：缓存控制
  - private 只允许客户端缓存结果，中间代理不允许缓存结果
  - public 中间代理也可以缓存结果
- Connection: 连接状态 keep-Alive 保持连接　close 关闭连接
- **content-Type**: 响应体的数据类型和字符集
- Date:响应时间
- Expires:缓存的失效时间
- Server :服务器的相关信息
- Set-Cookie: 设置cookie信息
- Strict-Transport-Security：强制要求客户端浏览器发送请求时，用https协议发送
- Traceid:跟踪id，是一个编号
- X-Ua-Compatible: 设置ie浏览器解析页面时，使用最新本的浏览器访问
- Content-Length:响应体的内容长度（单位为字节）

响应体

相应体的格式类型是非常灵活的，常见的响应格式有：

- HTML/JS/CSS
- 图片/音视频
- JSON

```
HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 21 Mar 2018 13:13:13 GMT
```

```
ETag: W/"a9-16248b12b64"
Content-Type: text/html; charset=UTF-8
Content-Length: 169
Date: Thu, 22 Mar 2018 12:58:41 GMT
Connection: keep-alive

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>首页</title>
</head>
<body>
  <h1>网站首页</h1>
</body>
</html>
```

- HTTP/1.1 200 OK：协议是HTTP 1.1版本，请求响应成功
- X-Powered-By: Express：自定义的头，表示用的框架，一般不返回容易造成安全漏洞。
- Accept-Ranges: bytes：告诉浏览器支持多线程下载
- Cache-Control: public, max-age=0：强制对所有静态资产进行缓存，即使它通常不可缓存。max-age指定多久缓存一次
- Last-Modified: Wed, 21 Mar 2018 13:13:13 GMT：这个资源最后一次被修改的日期和时间
- ETag: W/"a9-16248b12b64"：请求资源的标记/ID
- Content-Type: text/html; charset=UTF-8：返回响应体资源类型
- Content-Length: 169：响应体的长度
- Date: Thu, 22 Mar 2018 12:58:41 GMT：提供了日期的时间标志，标明响应报文是什么时间创建的

用nodejs搭建服务

```
// 1. 引入http模块
const http = require('http');
// 引入URL内置模块
const url = require('url')
// 引入 querystring模块，该模块的作用就是为了解析查询字符串
const qs = require('querystring')
// 2. 调用函数返回一个服务对象
// request 是对请求报文的封装对象 通过request对象可以获取到请求报文中的内容
// response 是对响应报文的封装对象 通过response对象可以设置http响应报文
// 每一个http请求到来后，都由回调函数处理请求，并设置响应
const server = http.createServer(function(request,response){
    // 获取请求类型
    console.log(request.method);
    // 获取请求的URL，返回的URL包含路径和查询字符串
    console.log(request.url);
    // 获取请求头的内容
    console.log(request.headers);
    // 注意这儿因为有短横岗，所以要用[]的方式访问
    console.log(request.headers['user-agent']);
    // 调动URL的parse方法自动解析URL
    console.log(url.parse(request.url,true));
    // 获取http请求中的版本号，很少用
    console.log(request.httpVersion);
```

```
    // 设置相响应头
response.setHeader('Content-type','text/html;charset=utf-8')
    // 获取请求体内容
    // 1. 声明变量
    let body = '';
    request.on('data',chunk =>{
        body + chunk
    })
    request.on('end',() =>{
        console.log(qs.parse(body))
        response.end('Over');
    })
        // 响应的数值
        // 1. 响应状态码的设置
        response.statusCode = 200;
        // 2. 响应字符串的数值 很少设置
        response.statusMessage = 'OK'
        // 3.相应头的数值,头的名字与值不能使用中文
        response.setHeader('name', 'xiaozuo')
        // 4. 相应体的设置
        //直接调用end方法，只能调用一次
        response.end('Over');
        // 调用write方法与end方法.write是流式写入，必须配合end结束流事件
        response.write('ddddg')
        response.write('vvvvv')
        response.end()

})
// 3.监听一个端口，启动服务
// 端口号：计算机的服务窗口号，总共65536个
// http的默认端口是80
server.listen(80,() =>{
    console.log('服务已经启动。。。。。。');
})
```

综合练习

```
// 判断输入的路径，返回不同的值，并判断bg的参数，根据参数设置页面的背景颜色
const url = require('url');
require('http')
.createServer(function (request, response) {
    const remethod = request.method;
    const reurl = url.parse(request.url, true);
    const repath = reurl.pathname;
    let bg = url.parse(request.url, true).query.bg;
    bg = bg || '#999'
    response.setHeader('Content-type', 'text/html;charset=utf-8')
    response.write(`<html style="background:${bg}"></html>`)
    if (remethod == 'GET' && repath == '/login') { //注意在路径的最前行方不能出现.符号
        response.end('登陆页面')
    } else if (remethod == 'GET' && repath == '/register') {
        response.end('注册页面')
    } else if (remethod == 'GET' && repath == '/home') {
        response.end('网站首页')
    } else if (remethod == 'GET' && repath == '/center') {
        response.end('个人中心页面')
    } else {
```

```
        response.end('404 未找到页面')
    }
}).listen(80, () => {
    console.log('服务已启动，正在监听80端口');
})
```

附录

响应状态码

响应状态码是服务器对结果的标识，常见的状态码有以下几种：

- 200：请求成功，浏览器会把响应体内容（通常是html）显示在浏览器中；
- 301：重定向，被请求的旧资源永久移除了（不可以访问了），将会跳转到一个新资源，搜索引擎在抓取新内容的同时也将旧的网址替换为重定向之后的网址；
- 302：重定向，被请求的旧资源还在（仍然可以访问），但会临时跳转到一个新资源，搜索引擎会抓取新的内容而保存旧的网址；
- 304：（Not Modified）请求资源未被修改，浏览器将会读取缓存；
- 403：forbidden 禁止的
- 404：请求的资源没有找到，说明客户端错误的请求了不存在的资源；
- 500：请求资源找到了，但服务器内部出现了错误；

http://127.0.0.1/是指向本机的ip地址，也可以使用localhost标识，她也是指向本机的。
ip是计算机在网络中的唯一地址

debug

```
Error:listen EADDRINUSE:address already in use :::8000
```

以上错误表示端口被占用
解决方法
ctrl +c 终止服务，在重启

dns 解析

浏览器发送请求时，会先进行dns解析，请求dns服务器查询当前域名对应的ip地址。

谷歌默认行为

favicon.ico 用于获取当前网站的小图标

Sec-Fetch-* 请求头

https://www.w3.org/TR/fetch-metadata/#sec-fetch-mode-header