入门HTTP协议看这篇文章就够了 - 爬虫和Web开发必备！

麦叔 [麦叔编程](javascript:void(0);)

<https://mp.weixin.qq.com/s/5zkMuUNMzugyfxNtg1VkIQ>

2021年04月16日 08:21

本文是麦叔爬虫训练营的餐前甜点**极速入门爬虫**的第1部分，后面两部分分别是：2. HTML, XPATH和JSON入门，3. 爬虫实战小项目。如果阅读量好，我会发后面两部分，所以请帮我点赞，转发，点再看。

HTTP对于爬虫是**重中之重**，对于学Web开发（前端或者后端）也是**必备技能**。本文涵盖了HTTP的必要知识点，把本文看透，基本够用了，然后再按需适当扩展就可以了。

另外，文末也有视频本教程的视频链接。

**1. 原理介绍**

爬虫就是用程序模拟浏览器的行为，发送请求给服务器，获取网页的内容，解析网页数据。

要学会爬虫，先要了解浏览器是如何和服务器交流的。浏览器通过HTTP协议和服务器交流。

image-20210410061755818

**2. HTTP协议简介**

**简介**

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写，主要用于网页的传输，现在也常应用网络API的开发（Restful API)。

HTTP是一个TCP/IP通信协议的最上层的协议之一（HTML 文件, 图片文件, 查询结果等）。TCP/IP协议有兴趣的可以自行学习，做Web开发的需要学习一下，对于爬虫不是必要内容。

以访问百度首页为例，可以看到网址的最前面就是http。

这里是https，你可以暂时忽略后面的s，https相对于http添加了数字证书加密功能，这不影响正常的爬虫。但写爬取的时候也要添加s，现在大部分网站都是https的。

了解https，请参考麦叔的文章，链接见底部第一篇。

image-20210410063113488

**基本工作原理**

HTTP是浏览器或者其他客户端（如手机App）和网站服务器之间沟通的协议。

浏览器作为HTTP**客户端**通过**URL**向HTTP服务端即**WEB服务器**发送所有请求。

常用的Web服务器有：**Nginx**，**Apache**，**IIS**服务器（微软的产品）等。

Web服务器接收到的请求后，向客户端发送响应信息。

HTTP默认端口号为80，但是你也可以改为8080或者其他端口。

**HTTP三个要点**

* **HTTP是无连接：** 无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
* **HTTP是媒体独立的：** 这意味着，只要客户端和服务器知道如何处理的数据内容，任何类型的数据都可以通过HTTP发送。客户端以及服务器指定使用适合的MIME-type内容类型。
* **HTTP是无状态：** HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

**3. chrome浏览器和开发者工具**

这个课程我们要使用chrome浏览器和它自带的开发者工具。

我们可以使用chrome来查看http的请求信息，基本操作步骤如下：

* 在网页上点右键，然后点检查：

image-20210410064218386

* 就会弹出开发者工具界面

image-20210410064444381

* 通过Elements我们可以看到网页的代码；通过Network我们可以看到HTTP的请求信息

**4. HTTP的消息结构**

**主要流程和概念**

* 请求
  + **地址**和**动词**：例如 **GET https://www.baidu.com**
  + **请求头（header)**：用来描述请求和发送者的一些信息
  + **请求参数**：以百度为例，要搜索的关键词。
* 相应
  + **响应代码**：200表示成功，404表示不存在等
  + **响应头header**：描述相应内容的一些信息
  + **响应内容**：HTML, JSON, 图片等

**请求和响应例子**

客户端请求：

GET /hello.txt HTTP/1.1  
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3  
Host: www.example.com  
Accept-Language: en, mi

服务端响应:

HTTP/1.1 200 OK  
Date: Mon, 27 Jul 2009 12:28:53 GMT  
Server: Apache  
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT  
ETag: "34aa387-d-1568eb00"  
Accept-Ranges: bytes  
Content-Length: 51  
Vary: Accept-Encoding  
Content-Type: text/plain

响应内容：

Hello World! My payload includes a trailing CRLF.

img

**一个很好用看请求的网站 httpbin.org/get**

通过这个网站可以更简单的查看请求的详细信息。

可以通过Python代码模拟各种请求参数和header，发送给这个网站，再来验证发送的是否正确。

image-20210410132123018

**5. 完整的网页请求过程**

注意你看到一个网页，通常是分多次请求的，一般的过程为：

* 先请求网页文本，也就是HTML
* 根据HTML中的指定的地址，请求其他内容，最常见的是：样式表，JavaScript，图片等

所以在chrome中可以看到很多个请求。如下图所示：

* 每一行都是一个请求
* 浏览器先请求www.baidu.com获得了HTML
* 然后发起了多个其他请求去获取css, woff2, png等其他资源

image-20210410072553801

这种特点给爬虫带来困难：

* 我们用代码发送请求，一行代码只能发送一次请求，也就是获得了最基本的HTML。
* 有的网站在第一次的HTML中没有包含真实的数据，真实的数据是通过后续请求获得的，所以有时候你用代码去请求发现获得的网页没有内容。

解决这个问题办法通常有两个：

* 分析各个请求，找到关键的后续请求，用代码发送后续关键请求获取数据。通常，我们没必要像浏览器一样发送所有后续请求，找到关键请求就够了。
* 使用selenium等浏览器驱动，模拟真实的浏览器操作，发送后续所有请求。

**6. 请求**

**1) 请求行**

下面例子中的第一行就是请求行，它包含几个关键信息：

* 请求方法：例子中是GET，表示你要干什么
* 请求的资源地址：例子中是 /hello.txt，结合下面的**Host** (ww.example.com)就可以拼接处完整的地址
* 使用的HTTP协议的版本号

HTTP协议有多个版本，现在主要用1.1，HTTP2也有一些应用

GET /hello.txt HTTP/1.1  
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3  
Host: www.example.com  
Accept-Language: en, mi

**2) 请求方法**

根据 HTTP 标准，HTTP 请求可以使用多种请求方法，这些方法表示这个请求要做什么。

HTTP1.0 定义了三种请求方法：GET, POST 和 HEAD方法。

HTTP1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法。

最常用的请求方法是：

* **GET** 请求表示要获取内容，它不会修改服务器上的数据
* **POST** 请求表示要提交内容，通常会修改服务器上的内容

这是最基本应该掌握的两个点。下面是完整的HTTP请求方法。

| **序号** | **方法** | **描述** |
| --- | --- | --- |
| 1 | **GET** | 请求指定的页面信息，并返回实体主体。 |
| 2 | HEAD | 类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头 |
| 3 | **POST** | 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。 |
| 4 | PUT | 从客户端向服务器传送的数据取代指定的文档的内容。 |
| 5 | DELETE | 请求服务器删除指定的页面。 |
| 6 | CONNECT | HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。 |
| 7 | OPTIONS | 允许客户端查看服务器的性能。 |
| 8 | TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |
| 9 | PATCH | 是对 PUT 方法的补充，用来对已知资源进行局部更新 。 |

**3) 请求参数**

**GET请求的参数**

**GET** 请求的参数都是放在URL中的，网页看见：

image-20210410065802606

请求参数是以键值对的形式出现，一般形式为：

* 网址的后面紧跟着是一个问号
* 问号的后面是一对对参数，形式为：参数1=参数1的值
* 多个参数之间用&符号隔开

举例：https://www.example.com/page1?参数1=参数1的值&参数2=参数2的值

看一下，上面的截图中有哪些参数，它们的值分别是什么？

**POST请求的参数**

* POST请求的参数放在请求体中
* 网页上看不见
* 但可以通过chrome开发者工具或者抓包工具看见

格式举例：

GET /hello.txt HTTP/1.1  
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3  
Host: www.example.com  
Accept-Language: en, mi  
  
{'name':'麦叔', '特征':'帅'，'坐标':'一个美丽的城市'}

**4) 请求头**

这部分内容非常重要，这是爬虫的关键技术之一，因为服务器通常通过请求头来：

* 判定请求是否合法，这是基本的反爬措施。爬虫通常需要添加多个请求头来模拟真实的网页请求
* 判断用户是否登陆了，一般通过**Cookie**这个特殊的请求头的内容来判定，只要我们给出合理的**Cookie**，服务器就认为我们是登录过了

下面是一个请求头列表，其中最重要的几个是：

* **User-Agent** 代表发起访问是什么浏览器。如果不写，基本会被判定为爬虫，直接被拒了。
* **Cookie** 里面记录了登录信息，或者上次请求服务端设置的信息，也是常用的反爬判定点
* **Referer** 表示这次请求是从哪里点过来的，有的网站不允许你直接访问某个网页，必须是从它的其他网页点过来才行，这时候就要设置一个Referer值，模拟是从别的网页点过来的情况

| **Header** | **解释** | **示例** |
| --- | --- | --- |
| Accept | 指定客户端能够接收的内容类型 | Accept: text/plain, text/html |
| Accept-Charset | 浏览器可以接受的字符编码集。 | Accept-Charset: iso-8859-5 |
| Accept-Encoding | 指定浏览器可以支持的web服务器返回内容压缩编码类型。 | Accept-Encoding: compress, gzip |
| Accept-Language | 浏览器可接受的语言 | Accept-Language: en,zh |
| Accept-Ranges | 可以请求网页实体的一个或者多个子范围字段 | Accept-Ranges: bytes |
| Authorization | HTTP授权的授权证书 | Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ== |
| Cache-Control | 指定请求和响应遵循的缓存机制 | Cache-Control: no-cache |
| Connection | 表示是否需要持久连接。（HTTP 1.1默认进行持久连接） | Connection: close |
| **Cookie** | HTTP请求发送时，会把保存在该请求域名下的所有cookie值一起发送给web服务器。 | Cookie: $Version=1; Skin=new; |
| Content-Length | 请求的内容长度 | Content-Length: 348 |
| Content-Type | 请求的与实体对应的MIME信息 | Content-Type: application/x-www-form-urlencoded |
| Date | 请求发送的日期和时间 | Date: Tue, 15 Nov 2010 08:12:31 GMT |
| Expect | 请求的特定的服务器行为 | Expect: 100-continue |
| From | 发出请求的用户的Email | From: user@email.com |
| Host | 指定请求的服务器的域名和端口号 | Host: www.zcmhi.com |
| If-Match | 只有请求内容与实体相匹配才有效 | If-Match: “737060cd8c284d8af7ad3082f209582d” |
| If-Modified-Since | 如果请求的部分在指定时间之后被修改则请求成功，未被修改则返回304代码 | If-Modified-Since: Sat, 29 Oct 2010 19:43:31 GMT |
| If-None-Match | 如果内容未改变返回304代码，参数为服务器先前发送的Etag，与服务器回应的Etag比较判断是否改变 | If-None-Match: “737060cd8c284d8af7ad3082f209582d” |
| If-Range | 如果实体未改变，服务器发送客户端丢失的部分，否则发送整个实体。参数也为Etag | If-Range: “737060cd8c284d8af7ad3082f209582d” |
| If-Unmodified-Since | 只在实体在指定时间之后未被修改才请求成功 | If-Unmodified-Since: Sat, 29 Oct 2010 19:43:31 GMT |
| Max-Forwards | 限制信息通过代理和网关传送的时间 | Max-Forwards: 10 |
| Pragma | 用来包含实现特定的指令 | Pragma: no-cache |
| Proxy-Authorization | 连接到代理的授权证书 | Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ== |
| Range | 只请求实体的一部分，指定范围 | Range: bytes=500-999 |
| **Referer** | 先前网页的地址，当前请求网页紧随其后,即来路 | Referer: http://www.zcmhi.com/archives/71.html |
| TE | 客户端愿意接受的传输编码，并通知服务器接受接受尾加头信息 | TE: trailers,deflate;q=0.5 |
| Upgrade | 向服务器指定某种传输协议以便服务器进行转换（如果支持） | Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11 |
| **User-Agent** | User-Agent的内容包含发出请求的用户信息 | User-Agent: Mozilla/5.0 (Linux; X11) |
| Via | 通知中间网关或代理服务器地址，通信协议 | Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1) |
| Warning | 关于消息实体的警告信息 | Warn: 199 Miscellaneous warning |

**7. 响应**

**1) 状态码**

当浏览者访问一个网页时，浏览者的浏览器会向网页所在服务器发出请求。当浏览器接收并显示网页前，此网页所在的服务器会返回一个包含HTTP状态码的信息头（server header）用以响应浏览器的请求。

HTTP状态码的英文为HTTP Status Code。

下面是常见的HTTP状态码：

* 200 - 请求成功
* 301 - 资源（网页等）被永久转移到其它URL
* 404 - 请求的资源（网页等）不存在
* 500 - 内部服务器错误

HTTP状态码由三个数字组成，第一个数字定义了状态码的类型，后两个数字表示具体的状态。HTTP状态码共分为5种类型：

| **分类** | **分类描述** |
| --- | --- |
| 1\*\* | 信息，服务器收到请求，需要请求者继续执行操作 |
| 2\*\* | 成功，操作被成功接收并处理 |
| 3\*\* | 重定向，需要进一步的操作以完成请求 |
| 4\*\* | 客户端错误，请求包含语法错误或无法完成请求 |
| 5\*\* | 服务器错误，服务器在处理请求的过程中发生了错误 |

HTTP状态码列表:

| **状态码** | **状态码英文名称** | **中文描述** |
| --- | --- | --- |
| 100 | Continue | 继续。客户端应继续其请求 |
| 101 | Switching Protocols | 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议 |
|  |  |  |
| 200 | OK | 请求成功。一般用于GET与POST请求 |
| 201 | Created | 已创建。成功请求并创建了新的资源 |
| 202 | Accepted | 已接受。已经接受请求，但未处理完成 |
| 203 | Non-Authoritative Information | 非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本 |
| 204 | No Content | 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档 |
| 205 | Reset Content | 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域 |
| 206 | Partial Content | 部分内容。服务器成功处理了部分GET请求 |
|  |  |  |
| 300 | Multiple Choices | 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择 |
| 301 | Moved Permanently | 永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替 |
| 302 | Found | 临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI |
| 303 | See Other | 查看其它地址。与301类似。使用GET和POST请求查看 |
| 304 | Not Modified | 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源 |
| 305 | Use Proxy | 使用代理。所请求的资源必须通过代理访问 |
| 306 | Unused | 已经被废弃的HTTP状态码 |
| 307 | Temporary Redirect | 临时重定向。与302类似。使用GET请求重定向 |
|  |  |  |
| 400 | Bad Request | 客户端请求的语法错误，服务器无法理解 |
| 401 | Unauthorized | 请求要求用户的身份认证 |
| 402 | Payment Required | 保留，将来使用 |
| 403 | Forbidden | 服务器理解请求客户端的请求，但是拒绝执行此请求 |
| 404 | Not Found | 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面 |
| 405 | Method Not Allowed | 客户端请求中的方法被禁止 |
| 406 | Not Acceptable | 服务器无法根据客户端请求的内容特性完成请求 |
| 407 | Proxy Authentication Required | 请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权 |
| 408 | Request Time-out | 服务器等待客户端发送的请求时间过长，超时 |
| 409 | Conflict | 服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突 |
| 410 | Gone | 客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置 |
| 411 | Length Required | 服务器无法处理客户端发送的不带Content-Length的请求信息 |
| 412 | Precondition Failed | 客户端请求信息的先决条件错误 |
| 413 | Request Entity Too Large | 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息 |
| 414 | Request-URI Too Large | 请求的URI过长（URI通常为网址），服务器无法处理 |
| 415 | Unsupported Media Type | 服务器无法处理请求附带的媒体格式 |
| 416 | Requested range not satisfiable | 客户端请求的范围无效 |
| 417 | Expectation Failed | 服务器无法满足Expect的请求头信息 |
|  |  |  |
| 500 | Internal Server Error | 服务器内部错误，无法完成请求 |
| 501 | Not Implemented | 服务器不支持请求的功能，无法完成请求 |
| 502 | Bad Gateway | 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应 |
| 503 | Service Unavailable | 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中 |
| 504 | Gateway Time-out | 充当网关或代理的服务器，未及时从远端服务器获取请求 |
| 505 | HTTP Version not supported | 服务器不支持请求的HTTP协议的版本，无法完成处理 |

**2) 响应头**

HTTP响应头包含了网页的重要描述信息，比如网页的格式，网页的过期时间等。

注意下面的**Set-Cookie**，它会设置网页的Cookie，在后面的请求中发送给服务器端，用来做身份判定或者反爬。

| **应答头** | **说明** |
| --- | --- |
| Allow | 服务器支持哪些请求方法（如GET、POST等）。 |
| **Content-Encoding** | 文档的编码（Encode）方法。只有在解码之后才可以得到Content-Type头指定的内容类型。利用gzip压缩文档能够显著地减少HTML文档的下载时间。Java的GZIPOutputStream可以很方便地进行gzip压缩，但只有Unix上的Netscape和Windows上的IE 4、IE 5才支持它。因此，Servlet应该通过查看Accept-Encoding头（即request.getHeader("Accept-Encoding")）检查浏览器是否支持gzip，为支持gzip的浏览器返回经gzip压缩的HTML页面，为其他浏览器返回普通页面。 |
| Content-Length | 表示内容长度。只有当浏览器使用持久HTTP连接时才需要这个数据。如果你想要利用持久连接的优势，可以把输出文档写入 ByteArrayOutputStream，完成后查看其大小，然后把该值放入Content-Length头，最后通过byteArrayStream.writeTo(response.getOutputStream()发送内容。 |
| Content-Type | 表示后面的文档属于什么MIME类型。Servlet默认为text/plain，但通常需要显式地指定为text/html。由于经常要设置Content-Type，因此HttpServletResponse提供了一个专用的方法setContentType。 |
| Date | 当前的GMT时间。你可以用setDateHeader来设置这个头以避免转换时间格式的麻烦。 |
| Expires | 应该在什么时候认为文档已经过期，从而不再缓存它？ |
| Last-Modified | 文档的最后改动时间。客户可以通过If-Modified-Since请求头提供一个日期，该请求将被视为一个条件GET，只有改动时间迟于指定时间的文档才会返回，否则返回一个304（Not Modified）状态。Last-Modified也可用setDateHeader方法来设置。 |
| Location | 表示客户应当到哪里去提取文档。Location通常不是直接设置的，而是通过HttpServletResponse的sendRedirect方法，该方法同时设置状态代码为302。 |
| Refresh | 表示浏览器应该在多少时间之后刷新文档，以秒计。除了刷新当前文档之外，你还可以通过setHeader("Refresh", "5; URL=http://host/path")让浏览器读取指定的页面。注意这种功能通常是通过设置HTML页面HEAD区的＜META HTTP-EQUIV="Refresh" CONTENT="5;URL=http://host/path"＞实现，这是因为，自动刷新或重定向对于那些不能使用CGI或Servlet的HTML编写者十分重要。但是，对于Servlet来说，直接设置Refresh头更加方便。 注意Refresh的意义是"N秒之后刷新本页面或访问指定页面"，而不是"每隔N秒刷新本页面或访问指定页面"。因此，连续刷新要求每次都发送一个Refresh头，而发送204状态代码则可以阻止浏览器继续刷新，不管是使用Refresh头还是＜META HTTP-EQUIV="Refresh" ...＞。 注意Refresh头不属于HTTP 1.1正式规范的一部分，而是一个扩展，但Netscape和IE都支持它。 |
| Server | 服务器名字。Servlet一般不设置这个值，而是由Web服务器自己设置。 |
| Set-Cookie | 设置和页面关联的Cookie。Servlet不应使用response.setHeader("Set-Cookie", ...)，而是应使用HttpServletResponse提供的专用方法addCookie。参见下文有关Cookie设置的讨论。 |
| WWW-Authenticate | 客户应该在Authorization头中提供什么类型的授权信息？在包含401（Unauthorized）状态行的应答中这个头是必需的。例如，response.setHeader("WWW-Authenticate", "BASIC realm=＼"executives＼"")。注意Servlet一般不进行这方面的处理，而是让Web服务器的专门机制来控制受密码保护页面的访问（例如.htaccess）。 |

**3) 响应内容和content type**

我们最常见的网页的内容格式为：

* HTML - 给人看的页面
* JSON - 数据格式，一般是API，给程序用的，比如手机APP
* XML - 也是数据格式

其实除此之外，还有图片，样式表，JavaScript等。

Content-Type就表示了网页的内容类型，它告诉浏览器文件类型，网页编码，帮助浏览器决定将以什么形式、什么编码读取这个文件。

Content-Type 标头告诉客户端实际返回的内容的内容类型。

语法格式：

Content-Type: text/html; charset=utf-8  
Content-Type: multipart/form-data; boundary=something

实例：

img

常见的媒体格式类型如下：

* text/html ：HTML格式
* text/plain ：纯文本格式
* text/xml ：XML格式
* image/gif ：gif图片格式
* image/jpeg ：jpg图片格式
* image/png：png图片格式

以application开头的媒体格式类型：

* application/xhtml+xml ：XHTML格式
* application/xml：XML数据格式
* application/atom+xml ：Atom XML聚合格式
* application/json：JSON数据格式
* application/pdf：pdf格式
* application/msword ：Word文档格式
* application/octet-stream ：二进制流数据（如常见的文件下载）
* application/x-www-form-urlencoded ：中默认的encType，form表单数据被编码为key/value格式发送到服务器（表单默认的提交数据的格式）

另外一种常见的媒体格式是上传文件之时使用的：

* multipart/form-data ：需要在表单中进行文件上传时，就需要使用该格式