03 HTTP 协议通信原理与 HTML 基础入门

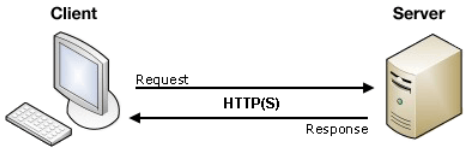
更新时间：2019-07-03 18:45:32

> 合理安排时间，就等于节约时间。
>
> ——培根

要编写网络爬虫就必须先了解 HTTP 通信协议是什么，它在我们浏览网页的时候是怎样运作的，它是如何构成的？除此之外还需要掌握网页(HTML)的结构以及怎样从网页中定位具体的元素。

HTTP 协议介绍

超文本传输协议（Hypertext Transfer Protocol，简称HTTP）是应用层协议。HTTP是一种请求/响应式的协议，即一个客户端与服务器建立连接后，向服务器发送一个请求；服务器接到请求后，给予相应的响应信息。



HTTP请求是一种人机可读明文，以百度为例，在浏览器上直接打开百度后可以从浏览器的开发者模式中看到以下网络请求内容：

```
GET http://www.baidu.com HTTP/1.1
```

通常 HTTP 消息包括客户机向服务器的请求消息和服务器向客户机的响应消息。这两种类型的消息由一个起始行、一个或者多个头域、一个只是头域结束的空行和可选的消息体组成。HTTP 的头域包括**通用头**、**请求头**、**响应头**和**实体头**四个部分。每个头域由一个域名、冒号（:）和域值三部分组成。域名大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

我并不打算将 HTTP 协议的所有内容直接罗列出来，协议说明和类手册一样无趣难懂，甚至会让你一下子失去对它的兴趣。虽然在将来你还是会仔细地阅读它，但我认为学习任何新的技术，在入手阶段最重要的是培养自己对这门技术的兴趣。因此，先不管具体的协议内容，只要知道HTTP请求就是向某一个服务器地址或者更准确地说是某一个具体的网络资源（URI）指定一种执行方法（HTTP 方法）并返回结果给我们就可以了。

`urllib`

为了能让枯燥的HTTP理论学习变得更为生动，作为程序员还是通过代码来一边动手一边学理论可能会更为深刻。那就借这个机会让我们来学习一个 Python 内置的http工具包urllib 详细的说明可以点击链接到 Python 的官网看，这个包不需要安装，直接在 Python 代码中引入就可以使用了。

```
import urllib
```

在下文的范例中我都将会用这个包来一边学习理论一边动手用 Python 来写代码。

HTTP的工作原理

HTTP协议采用请求/响应模型。客户端向服务器发送一个请求报文，服务器以一个状态作为响应。

以下是HTTP请求/响应的步骤：

- 客户端连接到 Web 服务器：HTTP 客户端与 web 服务器建立一个 TCP 连接。
- 客户端向服务器发起 HTTP 请求：通过已建立的 TCP 连接，客户端向服务器发送一个请求报文。
- 服务器接收 HTTP 请求并返回 HTTP 响应：服务器解析请求，定位请求资源，服务器将资源副本写到 TCP 连接，由客户端读取。
- 释放TCP连接：若 connection 模式为 close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接；若 connection 模式为 keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求。
- 客户端浏览器解析HTML内容：客户端将服务器响应的HTML文本解析并显示。

例如，在浏览器地址栏键入URL，按下回车键后会经历以下流程：

1. 浏览器向DNS服务器请求解析该URL中域名所对应的IP地址。
2. 解析出IP地址后，根据该IP地址和默认端口80，同服务器建立TCP连接。
3. 浏览器发出读取文件（URL中域名后面部分对应的文件）的HTTP请求，该请求报文作为TCP三次握手的第三个报文的数据发送给服务器。
4. 服务器对浏览器请求做出响应，并把对应的HTML文本发送给浏览器。
5. 释放TCP连接。
6. 浏览器将该HTML文本显示内容。

打开Chrome浏览器的开发者工具来查看请求的详情，如下所示：

```
×  Headers  Preview  Response  Cookies  Timing

▼ General
   Request URL: https://www.baidu.com/
   Request Method: GET
   Status Code: ● 200 OK
   Remote Address: 14.215.177.39:443
   Referrer Policy: no-referrer-when-downgrade

▶ Response Headers (20)
▼ Request Headers     view source
   Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,applicat
   ion/signed-exchange;v=b3
   Accept-Encoding: gzip, deflate, br
   Accept-Language: en-US,en;q=0.9,it;q=0.8,ja;q=0.7,zh-CN;q=0.6,zh;q=0.5,zh-TW;q=0.4,fr;q=0.3
   Cache-Control: max-age=0
   Connection: keep-alive
   Cookie: BIDUPSID=B8C9A78346AFD0C9A103479F3ECEFC63; PSTM=1524102899; BAIDUID=91B3B9BEB2F6C41AFAFEC977B8
   BEED37:FG=1; BDUSS=ZXNlJSTmV5bTdOR1lGbWdQTlE0aDFCaGRUME56dkNXbktRU3Nybk1DU05CaUpiQUFBQUFBJCQAAAAAAAA
   AAEAAAC5mLM1cmF5X29uX3JhaWxzAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAI15-
   lqNefpaV3; MCITY=-%3A; BDSFRCVID=bZLOJeCmHmJ6m539WHz_JIiPSgKK0gOTHlTr8P7U-YahRJ-VJeC6EG0Ptf8g0Ku-LKHr
   ogKK0gOTH6KF_2uxOjjg8UtVJeC6EG0P3J; H_BDCLCKID_SF=tR-joKthtKD3fJrYhPIV-PAt-U4Xat0XKKOLVM58JPOkeq8CDR_
   b35kshGtjXT5qKJTZoRCMyqonsxo2y5jHhpOy2hJ0JfQjaKJe-M3uyxTpsIJMKq_WbT8U5f5p2RQraKviaKOjBMb1MbcMe6Khj5cX
   jGu0q-LXKK_XWRTqa-cseJuk-PnVePulLpbZKxtqtJnqsRkhJD02obCG-6okDtuJXxbtQR5nWncKW-DE0DJRM5Cl-PQWhJtAX-R40
   5OT3T-O0KJcbR_a_nR-hPJvyTtsXnO7tPTTfJujVCKatCLbbP5kMtn_qttjMfbWetTbHD7yWCvyaRv5OR5Jj65K2M_HQfPfJpvLfI
   utaDjEtpABhb0C3MA--tRDQn3XKtnuyGn7hfQSbl_Msq0x0MOWe-bQypoa-nbJ0KOMahkb5h7xOKbF05C5DjoQeaA8qbbfb-oqBRr
   VaDIBbjrnhPF3DPbQKP6-35KH0RAeXPFbJKO2ohvd-6jY-tLUXfTrQl37JD6y-nPMLDb5jf523RjC0xtmXPoxJpOhMnbMopvaHl-M
   ERQvbURvDP-g3-AJaq8EtJAH_D8htC83fn5mbJbMqR_Lqxby26nHyeceaJ5nJDoWebQH5bDb0tCBM46Aa46qam7X-nvJQpP-hK5-b
   h5sQpobDJPfK4Qw3g5gKl0MLPjlbb0xynoD-fKi3fnMBMPe52OnaIb_LIFahKLxDj8-DjPW5ptXh4RtMD6XsJOOaCk5HqTRy4oTLn
   k1DpJGQ-4f3mTwL4TPMJvMsqjXW-6NBnT-btPeBjIqtR4f_Dt-f-tMq-ON-J5jMK6LjGAOetJyaR3ahqRvWJ5TDqnzy6jVjjJ-3P7
   EyMTA2GRg0hvctn3cShn13bKW05-AqJrKapjZHGcZ0l8K3l02V-bIe-t2ynQDX-R7btRMW20j0h7mWIQvsxA45J7cM4IseboJLfT-
   0bc4KKJxthF0HPonHjLhj6Q-3j; delPer=0; BD_CK_SAM=1; BD_HOME=1; ZD_ENTRY=google; BD_UPN=123253; BDRCVFR
   [feWj1Vr5u3D]=I67x6TjHwwYf0; PSINO=2; H_PS_PSSID=26522_1465_21124_28775_28722_28557_28835_28585_28641
   _28603_20718
   Host: www.baidu.com
   Upgrade-Insecure-Requests: 1
   User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like G
   ecko) Version/11.0 Mobile/15A372 Safari/604.1
```

HTTP请求 (Request)

HTTP请求的结构如下图所示：

\- 请求头部：请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头部通知服务器有关客户端请求的信息。

- 空行：最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。
- 请求包体：请求包体不在 GET 方法中使用，而是在 POST 方法中使用。POST 方法适用于需要客户填写表单的场合。与请求包体相关的最常使用的是包体类型Content-Type和包体长度Content-Length。

根据HTTP的请求原理我们尝试使用 urllib 来发起一个请求并读取返回的结果，代码如下所示：

```
# http-get.py
from urllib import request

response = request.urlopen('http://www.baidu.com')
print(response.read())
```

在命令行运行:

```
$ Python3 http-get.py
```

成功运行后你将会看到终端输出一大堆没有格式的HTML文本，`request.urlopen`默认向指定的url发出一个 get 请求，这也是大多网络工具包一种约定俗成的用法。`urlopen`函数返回的一个`response`对象，只要调用`read`方法就可以将响应对象中的正文内容读出。

参数化页面

不同的参数会直接影响网页的输出，这是服务端页面的特质。不过随着前端技术的发展，这一参数化的特性近年来也被引入到了不少知名的前端框架内，例如 ReactJS、AngularJS 和 Vue 等。参数化页面多用于`GET`方法，但实际上它没有被 HTTP 方法所限制，使用任何一种HTTP方法服务端都可以得到这些参数，因为这些参数就是 URL 的一部分。

查询字符串

所谓的**查询字符串**就是在URL后加上`?`，然后以`Key=Value&Key1=Value1&...&KeyN=valueN`的方式传递参数。这种方式传递的参数（长度）都会比较小，是最传统的一种参数传递方式。但由于这种形式的URL并不便于记忆，也违反了人机可读这一要求，在Web技术运用得比较好的一些网站和开发团队中已经慢慢被淘汰，或者说只作为某些特殊场合的一种补充。

下面是我在当当网查询关于"Python"书籍的URL:

```
http://e.dangdang.com/newsearchresult_page.html?keyword=Python
```

将`keyword=N`后的值换成不同的关键字就可以显示不同搜索结果，这是很多网站最经常采用的一种参数传递方式，当我们进行网页分析时必须留意。

下面的示例将在 Python 中发起一个带有查询字符串的url请求：

```
# http-querystring.py
from urllib import request
from urllib.parse import urlencode

url = 'http://e.dangdang.com/newsearchresult_page.html'
params = {'keyword': 'Python'}
query_string = urlencode(params)
url = url + "?" + query_string

with request.urlopen(url) as response:
    response_text = response.read()
    print(response_text)
```

上述代码首先用`url`定义了当当网的查询地址，然后用`params`定义一个字典变量承载查询参数，需要注意的是:

```
query_string = urlencode(params)
```

`urlencode`方法是是将`params`转化为一个具有UTF8编码方式（如果有多个参数`urlencode`会用`&`符号进行连接）的地址，最后将查询字符串与原URL合成最终的请求地址。

提交表单

表单通过`POST`或者`PUT`方法将网页上`<form>`元素中的输入域全部提交到`<form>`元素所指向的地址上，通过提交表单可以突破数据量小的局限，因为表单可以提交任何内容，包括各种二进制文件。但`POST`和`PUT`两种HTTP方法会给服务端带来较大的负荷，而且`POST`后的内容可能与开始访问的网页内容是完全不同的，这样爬虫就得进行额外的处理。这些内容会在下文的处理表单一节中用具体的应用示例说明，此处还是先将关注点移到HTTP请求上。

下面的示例用 Request 产生一个同时带有查询子符串的 URL 和用 GET 方法提交表单的 URL，然后再看看服务端会接收到什么，这里会将请求发到 http://httpbin.org/ 这个网站上，这是一个专门用来测试http协议的工具网站，当收到请求时就会将一些特殊的信息以json格式返回。

```
# http-post.py
from urllib import request, parse

data = parse.urlencode({'terms': 'Here is test'}).encode()
req = request.Request('http://httpbin.org/post?q=Python', data=data)
with request.urlopen(req) as response:
    print(response.read())
```

使用 POST 方法发起请求，大致上与之前的代码类似，与查询字符串不同的是 POST 请求需要将参数写入请求的正文中(body),所以在这里我们使用`data`参数来承载写入`body`的内容。

执行上述代码:

```
$ Python3 http-post.py
```

当代码成功执行时就会返回一个json字符串，为了方便阅读我将其格式化了一下，具体内容如下所示：

```
{
    "args": {
        "q": "Python"           # 查询字符串发向服务端的参数
    },
    "data": "",
    "files": {},
    "form": {
        "terms": "Here is test" # 写入正文通过POST方法发送的参数
    },
    "headers": {
        "Accept-Encoding": "identity",
        "Content-Length": "18",
        "Content-Type": "application/x-www-form-urlencoded",
        "Host": "httpbin.org",
        "User-Agent": "Python-urllib/3.7"
    },
    "json": null,
    "origin": "58.63.138.183, 58.63.138.183",
    "url": "https://httpbin.org/post?q=Python"
}
```

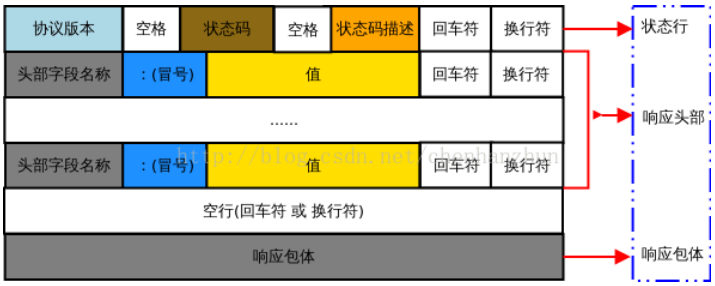

如果在服务端采用 Flask 或者 Django 这类 Python 服务端框架来接收以上请求，也会得到一个几乎与上述代码相同的服务端 Request 对象。通过这样一种穿透性的“透视”实验，我们可以很清楚地知道发出的请求最终在服务端会呈现什么。

> HTTP Request在发送到远程主机之前是会生成一种HTTP协议规定的文本，这里不会讲述原生的HTTP请求到底长什么样，因为爬虫根本不需要去了解它，我们只要在对象上知道请求怎么使用就够了。

查询字符串会被放到服务端请求对象的`args`数组里面（.NET会放置在QueryString对象中），而表单域中的数据则被放到`form`属性对象中。如果表单中带有`<input type="file" />`元素，则会将文件的内容放置到`files`数据组对象中。

HTTP响应 (Response)

如下图所示，HTTP响应的格式与请求的格式十分类似。



HTTP响应的协议格式代码如下：

```
<状态行>
<响应头>
<空行>
[<响应包体>]
```

状态行

状态行由HTTP协议版本字段、状态码和状态码的描述文本三个部分组成，它们之间使用空格隔开。

状态码由三位数字组成，第一位数字表示响应的类型，常用的状态码有五大类。

- 1xx：表示服务器已接收了客户端请求，客户端可继续发送请求。
- 2xx：表示服务器已成功接收到请求并进行处理。
- 3xx：表示服务器要求客户端重定向。
- 4xx：表示客户端的请求有非法内容。
- 5xx：表示服务器未能正常处理客户端的请求而出现意外错误。

状态码描述文本有如下取值。

200 OK：表示客户端请求成功；
400 Bad Request：表示客户端请求有语法错误，不能被服务器所理解。
401 Unauthonzed：表示请求未经授权，该状态代码必须与WWW-Authenticate报头域一起使用。
403 Forbidden：表示服务器收到请求，但是拒绝提供服务，通常会在响应正文中给出不提供服务的原因。
404 Not Found：请求的资源不存在，例如，输入了错误的URL。
500 Internal Server Error：表示服务器发生不可预期的错误，导致无法完成客户端的请求。
503 Service Unavailable：表示服务器当前不能够处理客户端的请求，在一段时间之后，服务器可能会恢复正常。

响应头部

响应头可能包括以下内容：

- Location：Location响应报头域用于重定向接收者到一个新的位置。例如，客户端请求的页面已不在原先的位置，为了让客户端重定向到这个页面新的位置，服务器端可以发回Location响应报头后使用重定向语句，让客户端去访问新的域名所对应的服务器上的资源。
- Server：Server响应报头域包含了服务器用来处理请求的软件信息及其版本。它和User-Agent请求报头域是相对应的，前者发送服务器端软件的信息，后者发送客户端软件（浏览器）和操作系统的信息。
- Vary：指示不可缓存的请求头列表。
- Connection：连接方式。
  对于请求来说：close（告诉Web服务器或者代理服务器，在完成本次请求的响应后断开连接，不等待本次连接的后续请求了）；keepalive（告诉Web服务器或者代理服务器，在完成本次请求的响应后保持连接，等待本次连接的后续请求）。

对于响应来说：close（连接已经关闭）；keepalive（连接保持，在等待本次连接的后续请求）。

- Keep-Alive：如果浏览器请求保持连接，则该头部表明希望Web服务器保持连接多长时间（秒）。例如，Keep-Alive：300。
- WWW-Authenticate：WWW-Authenticate响应报头域必须被包含在401（未授权的）响应消息中，这个报头域和前面讲到的Authorization请求报头域是相关的，当客户端收到401响应消息后，就要决定是否请求服务器对其进行验证。如果要求服务器对其进行验证，就可以发送一个包含了Authorization 报头域的请求。

空行：最后一个响应头部之后是一个空行，发送回车符和换行符，通知服务器以下不再有响应头部。

响应包体：服务器返回给客户端的文本信息。

正如你所见，在响应中唯一真正的区别在于第一行中用状态信息代替了请求信息。状态行（status line）通过提供一个状态码来说明所请求的资源情况。以下就是一个HTTP响应的例子：

```
HTTP/1.1 200 OK
Date: Sat, 31 Dec 2005 23:59:59 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 122
<html>
<head>
<title>Homepage</title>
</head>
<body>
<!-- body goes here -->
</body>
</html>
```

在状态行之后是一些首部。通常，服务器会返回一个名为 Data 的首部，用来说明响应生成的日期和时间（服务器通常还会返回一些关于其自身的信息，尽管并非是必需的）。接下来的两个首部大家应该很熟悉，就是与POST请求中一样的 Content-Type 和 Content-Length。在本例中，首部 Content-Type 指定了MIME类型HTML（text/html），其编码类型是ISO-8859-1（这是针对美国英语资源的编码标准）。

响应包体

响应包体所包含的就是所请求资源的 HTML 源文件（尽管还可能包含纯文本或其他资源类型的二进制数据）。浏览器将把这些数据显示给用户。

注意，这里并没有指明针对该响应的请求类型，不过这对于服务器来说并不重要。客户端知道每种类型的请求将返回什么类型的数据，并决定如何使用这些数据。

HTML 入门

> HTML 是一种相当简单的、由不同元素组成的标记语言，它可以应用于文本片段，使文本在文档中具有不同的含义（它是一个段落吗？它是一个项目列表吗？它是一个表格吗？），将文档结构化为逻辑块（文档是否有头部？有三列内容？有一个导航菜单？），并且可以将图片，影像等内容嵌入到页面中。
>
> —— MDN

```
<html>
    <head>
        <title>网页标题</title>
    </head>
    <body>
        <!--网页正文-->
    </body>
</html>
```

HTML说简单是由几个简单标记能构成的，说复杂是由于它有许许多多的标记，每种标记有着不同的显示效果与用法，并由此构成复杂 DOM 对象模型树，对于开发网络爬虫程序而言我们并不需要掌握全部的网络知识，而只要知道**网页是一个结构严格完成的，相互嵌套的、由标记组成的结构化文档**就可以了。毕竟我们只关心藏在这些结构中的哪些**数据**是最需要的，要怎么样去定位和将它们提取出来即可。

在网络爬虫中我们需要记住的标记有以下这些:

- `<a href="链接地址">显示的文字</a>` - 链接标记，这个标记是最常用的，因为它带有链接地址，是指引爬虫前进的**路标**；
- `<link href="链接地址" />` - `<link>`是一种非可视性的链接标记，可以链接各种不同的资源，常在`<head>`标记内用于引用样式表或脚本，某些时候也用于引用网页，属于网爬虫爬网时的**“线索”**导引；

你掌握的HTML知识越多对日后进行网页分析的效果就越好，网页分析就是读懂HTML网页代码，所以要进一步精研网络爬虫技术的话，经常去MDN补充学习HTML的相关知识是非常有必要的。

CSS 元素选择器 入门

CSS 是网页的**“画笔”**，它决定了网页的外观显示效果，所以被称为“样式表”。样式表的显示技术并不是网络爬虫最关注的技术，样式表中的元素选择与定位技术才是我们的重点。

要在结构复杂的网页中对某个或某些元素指定特定的样式就需要有准确的定位办法，这种定位的办法就是CSS选择器。

以下的这些CSS选择器都是我们在网络爬虫中经常使用到的：

- 简单选择器（Simple selectors）：通过元素类型、class 或 id 匹配一个或多个元素。
- 属性选择器（Attribute selectors）：通过 属性 / 属性值 匹配一个或多个元素。
- 伪类（Pseudo-classes）：匹配处于确定状态的一个或多个元素，比如被鼠标指针悬停的元素，当前被选中或未选中的复选框，或元素是DOM树中一父节点的第一个子节点。
- 伪元素（Pseudo-elements）:匹配处于相关的确定位置的一个或多个元素，例如每个段落的第一个字，或者某个元素之前生成的内容。
- 组合器（Combinators）：这里不仅仅是选择器本身，还有以有效的方式组合两个或更多的选择器，用于非常特定的选择方法。例如，你可以只选择divs的直系子节点的段落，或者直接跟在headings后面的段落。

> 建议先阅读上文中的每种选择器链接地址的内容，先建立一个选择器的具体概念。

接下来，我将通过代码来演示在Python中应该如何应用选择器。此时我们需要使用一个工具包pyQuery，它是一个可以提供像jQuery类似语法的Python工具包。

```
$ pip3 install pyQuery
```

以下代码是上文中对应的各种选择器的具体实现方式：

```
# coding:utf-8
from pyquery import PyQuery as pq

html = """
<html>
    <title></title>
    <body>
        <div item-prop="link">
            <a class="link" href="http://www.baidu.com">百度</a>
        </div>
        <ul>
            <li data-quantity="1kg" data-vegetable>番茄</li>
            <li data-quantity="3" data-vegetable>洋葱</li>
            <li data-quantity="3" data-vegetable>大蒜</li>
            <li data-quantity="700g" data-vegetable="not spicy like chili">红椒</li>
            <li data-quantity="2kg" data-meat>鸡肉</li>
            <li data-quantity="optional 150g" data-meat>玉米粒</li>
            <li data-quantity="optional 10ml" data-vegetable="liquid">橄榄油</li>
            <li data-quantity="25cl" data-vegetable="liquid">酒
                <i>图标</i>
            </li>
        </ul>
    </body>
</html>
"""

doc = pq(html) # 构造一个pyQuery实例对象并加载HTML的内容

# class 类选择器
link = doc(".link")
print('链接:%s' % link.text())  # 打印链接中的文字

# 属性选择器 - 选取所有蔬菜 带有 data-vegetable标记
items = doc("[data-vegetable]")
print('----------\n')
print('各种蔬菜\n')
items.each(lambda i: print(pq(this).text()))  # 打印全部蔬菜的内容
print('----------\n')

# 伪类选择器
pseudo = doc("li:first-child")  # 选择ul标记内的第一个li元素
print('选择ul标记内的第一个li元素:%s' % pseudo.text())

# 组合与多重选择器
combinator = doc("ul>li>i")
print(combinator.text())
```

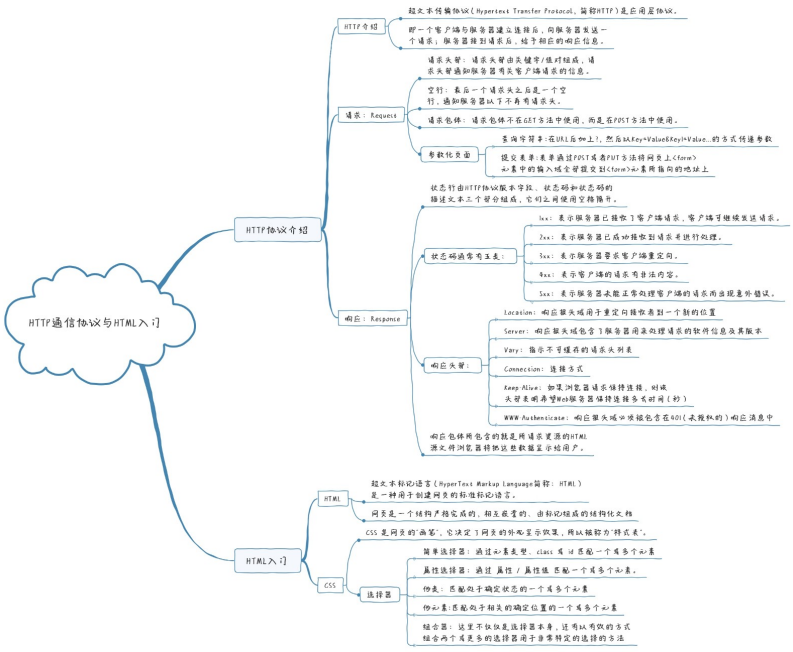

总结

我从事Web开发10多年用过很多语言，也用过很多框架，但这些语言、框架无论是前端、后端还是网络爬虫技术，最终还是离不开它们的的本质 —— HTTP。

所以你需要清楚地:

- 理解HTTP的通信息过程：请求->响应 模式
- 请求Request是怎么产生的，它包含什么内容。
- 响应是怎么得到的，里面哪些内容是我们所需要的。

还有就是最必要的HTML与CSS技术，这些内容都需要你持续地加深理解，这将会对后续的学习有极大的帮助。



02 网络爬虫基础知识准备及基本开发环境介绍

04 动手开发最简单的单文件爬虫