第 3 章

网络通信基础知识

使用网络爬虫爬取数据，不仅需要了解网页的结构，而且了解网页的请求原理也是非常有必要的。本章结合浏览网页的过程，介绍 HTTP 请求的原理和网络通信的基础知识。学习 Python 提供的用于网络编程和通信的各种模块，主要介绍 Python 的 Socket TCP 程序的开发。

视频讲解

3.1 网络协议

3.1.1 互联网 TCP/IP 协议

计算机为了联网就必须规定通信协议，早期的计算机网络都是由各厂商自己规定一套协议，IBM、Apple 和 Microsoft 都有各自的网络协议，互不兼容。这就好比一群人有的说英语，有的说法语，有的说德语，说同一种语言的人可以交流，不同的语言之间就不行了。

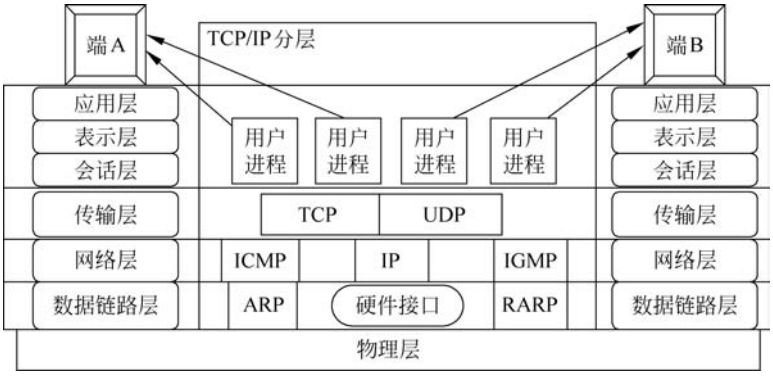

图 3-1 互联网协议

为了把全世界的所有不同类型的计算机都连接起来，就必须规定一套全球通用的协议，为了实现互联网这个目标，国际组织制定了 OSI 七层模型互联网协议标准，如图 3-1 所示。因为互联网协议包含了上百种协议标准，但是最重要的两个协议是 TCP 和 IP 协议，所以，大家把互联网的协议简称为 TCP/IP 协议。

3.1.2 IP 协议和端口

1. IP 协议

通信的时候，双方必须知道对方的标识，就像发邮件时必须知道对方的邮件地址。互联网中每个计算机的唯一标识就是 IP 地址，类似于 202.196. 32.7。如果一台计算机同时接入两个或更多的网络，如路由器，它就会有两个或多个 IP 地址。所以，IP 地址对应的实际上是计算机的网络接口，通常是网卡。

IP 协议负责把数据从一台计算机通过网络发送到另一台计算机。数据被分割成一小块一小块，然后通过 IP 包发送出去。由于互联网链路复杂，两台计算机之间经常有多条线路。因此，路由器就负责决定如何把一个 IP 包转发出去。IP 包的特点是按块发送，途经多个路由，但不保证能到达，也不保证顺序到达。

IP 地址实际上是一个 32 位整数(称为 IPv4)，以字符串表示的 IP 地址(如 192.168.0.1)实际上是把 32 位整数按 8 位分组后的数字表示，目的是便于阅读。

IPv6 地址实际上是一个 128 位整数，它是目前使用的 IPv4 的升级版，以字符串表示，类似于 2001:0db8:85a3:0042:1000:8a2e:0370:7334。

2. 端口

一个 IP 包除了包含要传输的数据外，还包含源 IP 地址和目标 IP 地址、源端口和目标端口。

端口有什么作用？在两台计算机通信时，只发 IP 地址是不够的，因为同一台计算机上运行着多个网络程序(如浏览器、QQ 等网络程序)。一个 IP 包来了之后，到底是交给浏览器还是 QQ，就需要端口号来区分。每个网络程序都向操作系统申请唯一的端口号，这样，两个进程在两台计算机之间建立网络连接就需要各自的 IP 地址和各自的端口号。例如，浏览器常常使用 80 端口，FTP 程序使用 21 端口，邮件收发使用 25 端口。

网络上两个计算机之间的数据通信，归根到底就是不同主机的进程交互，而每个主机的进程都对应着某个端口。也就是说，单独靠 IP 地址是无法完成通信的，必须有 IP 和端口。


视频讲解

3.1.3 TCP 和 UDP 协议

TCP 协议是建立在 IP 协议之上的。TCP 协议负责在两台计算机之间建立可靠连接，保证数据包按顺序到达。TCP 协议会通过握手建立连接，然后对每个 IP 包编号，确保对方按顺序收到，如果包丢掉了就自动重发。

许多常用的更高级的协议都是建立在 TCP 协议基础上的，如用于浏览器的 HTTP 协议、发送邮件的 SMTP 协议等。

UDP 协议同样是建立在 IP 协议之上的，但是 UDP 协议面向无连接的通信协议，不保

证数据包的顺利到达，不可靠传输，所以，效率比 TCP 要高。

使用 UDP 协议时，不需要建立连接，只需要知道对方的 IP 地址和端口号，就可以直接发数据包。但是，能不能到达就不知道了。虽然用 UDP 传输数据不可靠，但它的优点是速度快，对于不要求可靠到达的数据，就可以使用 UDP 协议。

3.1.4 HTTP 和 HTTPS 协议

HTTP 协议（超文本传输协议）被用于在 Web 浏览器和网站服务器之间传递信息，HTTP 协议以明文方式发送内容，不提供任何方式的数据加密。如果攻击者截取了 Web 浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息。因此，HTTP 协议不适合传输一些敏感信息，如信用卡号、密码等支付信息。

为了解决 HTTP 协议的这一缺陷，需要使用另一种协议：HTTPS（超文本传输安全协议）。为了数据传输的安全，HTTPS 在 HTTP 的基础上加入了 SSL 协议，SSL 依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

1. HTTP 协议的基本概念

HTTP 协议是一个基于请求与响应模式的、无状态的、应用层的协议，常基于 TCP 的连接方式。HTTP 1.1 版本中给出了一种持续连接的机制，绝大多数的 Web 开发，都是构建在 HTTP 协议之上的 Web 应用。

HTTP URL（URL 是一种特殊类型的 URI，包含了用于查找某个资源的足够的信息）的格式如下：

```
http://host[":"port][abs_path]
```

http 表示要通过 HTTP 协议来定位网络资源；host 表示合法的 Internet 主机域名或者 IP 地址；port 指定一个端口号，为空则使用默认端口 80；abs_path 指定请求资源的 URI，即网页文件的路径。

例如输入：http://www.zut.edu.cn/info/1043/22024.htm

www.zut.edu.cn 就是中原工学院主机 DNS 域名，使用默认端口 80，info/1043/22024.htm 是网页文件在服务器上的相对路径。

2. HTTP 网络请求

HTTP 请求消息由四部分组成，分别是请求行、请求头部、空行、请求数据。HTTP 请求消息一般格式结构如图 3-2 所示。

图 3-2 HTTP 请求消息一般格式

下面结合两个典型的 HTTP 请求实例，详细介绍 HTTP 请求信息的各个组成部分。实例内容如下：

1）GET 请求例子

```
GET /562f25980001b1b106000338.jpg HTTP/1.1
Host: img.mukewang.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/51.0.2704.106 Safari/537.36
Accept: image/webp,image/*,*/*;q=0.8
Referer: http://www.imooc.com/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
```

第一部分：请求行

请求行用来说明请求类型、要访问的资源及所使用的 HTTP 版本，如下所示：

```
GET /562f25980001b1b106000338.jpg HTTP/1.1
```

请求行以一个 GET 方法符号开头，以空格分开，后面跟着请求的 URI 和协议的版本 HTTP/1.1。

上述例子中，GET 说明请求类型为 GET 方法，“/562f25980001b1b106000338.jpg”为要访问的资源，该行的最后一部分说明使用的是 HTTP 1.1 版本。

客户端和服务器之间交互会使用不同的请求类型。HTTP 协议的请求方法有 GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。表 3-1 列出了 HTTP 请求报文的几种请求方法。

表 3-1　HTTP 请求报文的请求方法

方法（操作）	意　义
GET	请求获取由 URL 所标识的资源
HEAD	请求获取由 URL 所标识的资源的响应消息报头
POST	向服务器提交信息
PUT	在指明的 URL 下存储一个文档
DELETE	删除指明的 URL 所标签的资源
TRACE	用来进行环回测试的请求报文
OPTIONS	请求一些选项的信息
CONNECT	用于代理服务器

其中 GET 方法是最常见的一种请求方式。当客户端要从服务器中读取文档时，单击网页上的链接或者通过在浏览器的地址栏输入网址来浏览网页的，使用的都是 GET 方式。GET 方法要求服务器将 URL 定位的资源放在响应报文的数据部分，回送给客户端。使用 GET 方法时，请求参数和对应的值附加在 URL 后面，利用一个问号“?”代表 URL 的结尾与请求参数的开始，传递参数长度受限制。例如，/index.jsp?id=100&op=bind，这样通过 GET 方式传递的数据直接表示在 URL 网页地址中。

地址中“?”之后的部分就是通过 GET 发送的请求数据，用户可以在地址栏中清楚地看

到，各个数据之间用“&”符号隔开。显然，这种方式不适合传送私密数据。另外，由于不同的浏览器对地址的字符限制也有所不同，一般最多只能识别 1024 个字符。所以，如果需要传送大量数据，也不适合使用 GET 方式。

对于上面提到的不适合使用 GET 方式的情况，可以考虑使用 POST 方式，因为使用 POST 方法可以允许客户端给服务器提供信息较多。POST 方法将请求参数封装在 HTTP 请求数据中，以名称/值的形式出现，可以传输大量数据，这样 POST 方式对传送的数据大小没有限制，而且也不会显示在 URL 中。

还有 HEAD 方法，HEAD 方法就像 GET，只不过服务端接收到 HEAD 请求后只返回响应头，而不会发送响应内容。当只需要查看某个页面的状态时，使用 HEAD 是非常高效的，因为在传输的过程中省去了页面内容。

第二部分：请求头信息

从第二行起为请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息。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于客户端请求的信息，典型的请求头有如下 3 个：

- User-Agent：产生请求的浏览器类型。该信息由浏览器来定义，并且在每个请求中自动发送；
- Accept：客户端可识别的内容类型列表；
- Host：请求的主机名，将指出被请求网页资源的 Internet 主机和端口（即目的地）。

第三部分：空行

请求头部后面的空行是必需的。通知服务器以下不再有请求头。

即使第四部分的请求数据为空，也必须有空行。

第四部分：请求数据

请求数据也叫主体（请求包体），可以添加任意数据。

这个例子的 GET 请求数据为空。请求数据不在 GET 方法中使用，而是在 POST 方法中使用。

2）POST 请求例子

POST 方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是 Content-Type 和 Content-Length。

```
POST / HTTP1.1
Host:www.wrox.com
User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET
CLR 3.0.04506.648; .NET CLR 3.5.21022)
Content-Type:application/x-www-form-urlencoded
Content-Length:40
Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley
```

第一部分：请求行，第一行说明是 POST 请求，以及 HTTP 1.1 协议版本。

```
POST / HTTP1.1
```

第二部分：请求头部，第二行至第六行。

```
Host:www.wrox.com
User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET
CLR 3.0.04506.648; .NET CLR 3.5.21022)
Content-Type:application/x-www-form-urlencoded
Content-Length:40
Connection: Keep-Alive
```

第三部分：空行，请求数据前面的空行（第七行）。

第四部分：请求数据，第八行。

```
name=Professional%20Ajax&publisher=Wiley
```

3. HTTP 响应消息

服务器在接收和解释请求消息后，返回一个 HTTP 响应消息。HTTP 响应也由四个部分组成，分别是状态行、消息报头、空行和响应正文。下面是一个 HTTP 响应消息例子，如图 3-3 所示。

图 3-3　HTTP 响应消息例子

```
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>Wrox Homepage</title>
</head>
<body>
<!-- body goes here -->
</body>
</html>
```

第一部分：状态行，由 HTTP 协议版本号、服务器发回的响应状态代码、表示响应状态

的文本三部分组成。上述例子中，第一行为状态行，HTTP/1.1 表明 HTTP 版本为 1.1 版本，状态码为 200，状态消息为 OK。

其中，状态代码由三位数字组成，第一个数字定义了响应的类别，且有如下五种可能的取值：

- 1xx：指示信息，表示请求已接收，继续处理；
- 2xx：成功，表示请求已被成功接收、理解、接受；
- 3xx：重定向，要完成请求必须进行更进一步的操作；
- 4xx：客户端错误，请求有语法错误或请求无法实现；
- 5xx：服务器端错误，服务器未能实现合法的请求。

常见状态代码如表 3-2 所示。

表 3-2 常见状态代码

常见状态代码	状 态 描 述	说 明
200	OK	客户端请求成功
400	Bad Request	客户端请求有语法错误，不能被服务器所理解
401	Unauthorized	请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用
403	Forbidden	服务器收到请求，但是拒绝提供服务
404	Not Found	请求资源不存在，例如输入了错误的 URL
500	Internal Server Error	服务器发生不可预期的错误
503	Server Unavailable	服务器当前不能处理客户端的请求，一段时间后可能恢复正常

例如，HTTP/1.1 200 OK (CRLF)。

第二部分：消息报头，用来说明客户端要使用的一些附加信息。

例如：

- Date：生成响应的日期和时间；
- Content-Type：指定了 MIME 类型的 HTML(text/html)，编码类型是 ISO-8859-1；
- 第三部分：空行，消息报头后面的空行是必需的；
- 第四部分：响应正文，服务器返回给客户端的文本信息。

上例中空行后面的 html 部分为响应正文。

3.1.5 HTTP 基本原理与机制

1. HTTP 请求与响应机制

HTTP 协议用于客户端(如浏览器)与服务器之间的通信，如图 3-4 所示。在通信线路两端，必定一端是客户端，另一端是服务器。客户端与服务器的角色不是固定的，一端充当客户端，也可能在某次请求中充当服务器，这取决于请求的发起端。

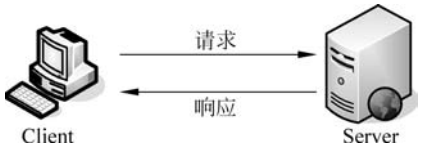

图 3-4 HTTP 请求与响应模式

HTTP 协议属于应用层，建立在传输层协议 TCP 之上。客户端通过与服务器建立 TCP 连接，之

后发送 HTTP 请求与接收 HTTP 响应都是通过访问 Socket 接口来调用 TCP 协议实现。

2. HTTP 请求/响应的步骤

HTTP 请求/响应的步骤如图 3-5 所示。

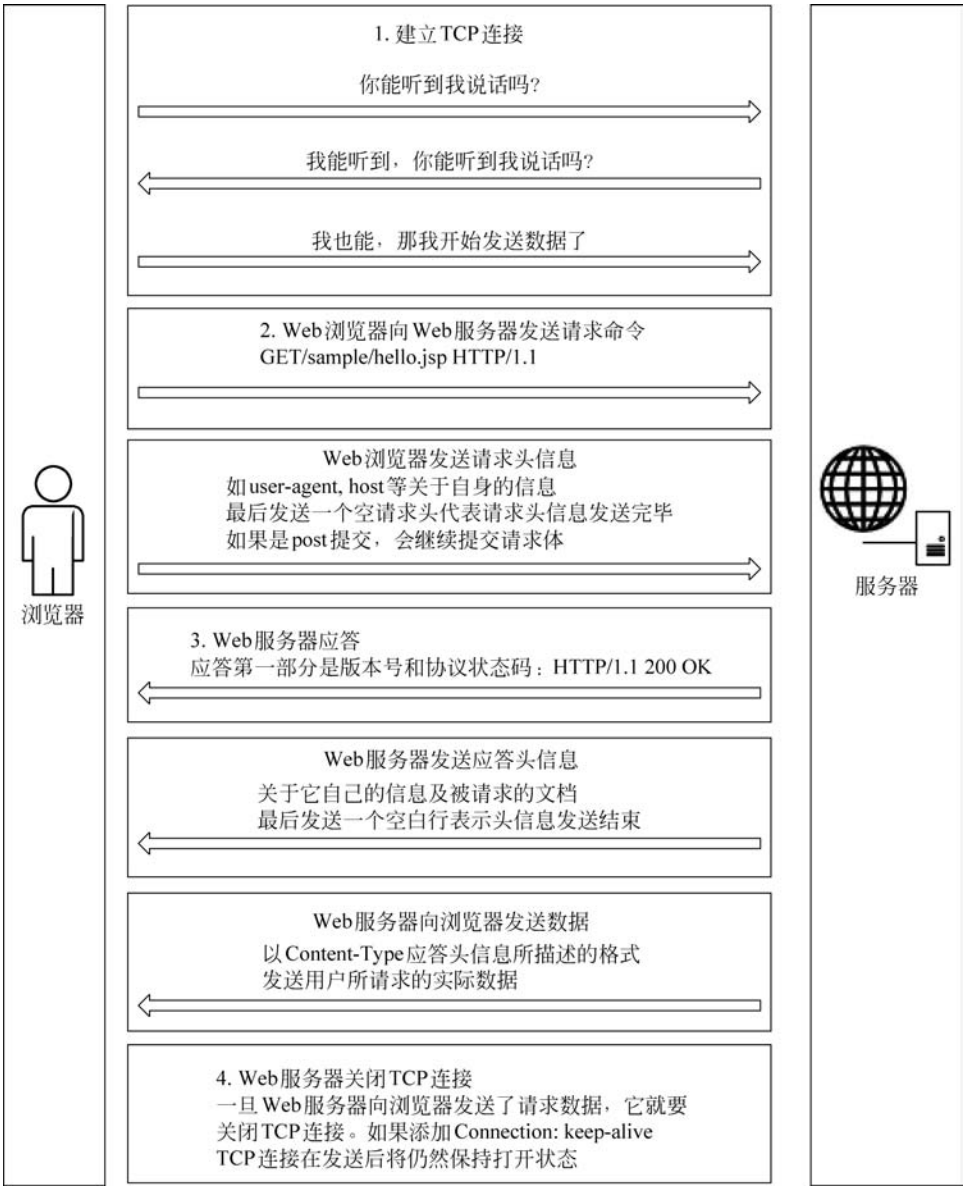

图 3-5　HTTP 请求/响应的步骤

1）建立 TCP 连接，客户端连接到 Web 服务器

一个 HTTP 客户端通常是 Web 浏览器，与 Web 服务器的 HTTP 端口（默认为 80）建立一个 TCP 连接。

在 HTTP 工作开始之前，Web 浏览器首先要通过网络与 Web 服务器建立连接，该连接是通过 TCP 来完成的，该协议与 IP 协议共同构建 Internet，即著名的 TCP/IP 协议族，因此

Internet 又被称作 TCP/IP 网络。

HTTP 是比 TCP 更高层次的应用层协议，根据规则，只有低层协议建立之后才能进行更高层协议的连接。因此，首先要建立 TCP 连接，一般 TCP 连接的端口号是 80。TCP 连接中比较熟悉的就是三次握手。

2）发送 HTTP 请求

通过 TCP 套接字连接，客户端向 Web 服务器发送一个文本的请求报文，一个请求报文由请求行、请求头信息、空行和请求数据 4 部分组成。例如，GET/sample/hello.jsp HTTP/1.1。

浏览器发送其请求命令(如 GET)之后，还要以请求头信息的形式向 Web 服务器发送一些别的信息，这些信息用来描述浏览器本身。之后浏览器发送了一空白行来通知服务器，表示它已经结束了该头信息的发送。若是 POST 请求，还会在发送完请求头信息之后发送请求体。

3）服务器接受请求并返回 HTTP 响应

Web 服务器解析请求，定位请求资源。服务器将资源副本写到 TCP 套接字，由客户端读取。一个响应由状态行、响应头信息、空行和响应数据 4 部分组成。

例如，浏览器向服务器发出请求后，服务器会向浏览器回送应答。

```
HTTP/1.1 200 OK
```

应答的第一部分是协议的版本号和应答状态码。

正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档，最后以一个空白行来表示响应头信息发送到此结束。

Web 服务器向浏览器发送响应头信息后，它就以 Content-Type 应答头信息所描述的格式发送用户所请求的实际数据。

4）释放连接 TCP 连接

一般情况下，一旦 Web 服务器向浏览器发送了请求数据，它就要关闭 TCP 连接。如果浏览器或者服务器在其头信息加入了如下这行代码：

```
Connection:keep-alive
```

TCP 连接在发送后将仍然保持打开状态，该连接会保持一段时间。于是浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

5）客户端浏览器解析 HTML 内容

客户端浏览器首先解析响应的状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的 HTML 文档和文档的字符集。客户端浏览器读取响应数据 HTML，根据 HTML 的语法对其进行格式化，并在浏览器窗口中显示。

例如，在浏览器地址栏键入 URL，按 Enter 键之后会经历以下流程。

(1) 浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址。

(2) 解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立 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 文件解析并显示内容。

3.1.6 使用 Fiddler 抓包验证请求信息和响应信息

Fiddler 是最强大最好用的 Web 调试工具之一，它能记录所有客户端和服务器的 HTTP 和 HTTPS 请求。允许用户监视、设置断点甚至修改输入/输出数据。Fiddler 包含了一个强大的基于事件脚本的子系统，并且能使用.NET 语言进行扩展。Fiddler 无论对开发人员或者测试人员来说，都是非常有用的工具。

1. Fiddler 的工作原理

Fiddler 是以代理 Web 服务器的形式工作的，它使用代理地址：127.0.0.1，端口：8888。当 Fiddler 退出的时候它会自动注销，这样就不会影响别的程序。不过如果 Fiddler 非正常退出，这时因为 Fiddler 没有自动注销，会造成网页无法访问。解决的办法是重新启动 Fiddler。

Fiddler 作为一个抓包工具，当浏览器访问服务器会形成一个请求，此时 Fiddler 就处于请求之间，当浏览器发送请求，会先经过 Fiddler，然后再到服务器；当服务器有返回数据给浏览器显示时，也会先经过 Fiddler，然后数据才到浏览器中显示。这样一个过程，Fiddler 就抓取到全部请求和响应信息。

2. 测试 GET 的请求和应答

首先构建一个 GET 请求，Fiddler 选择“组合器”选项卡，设置请求方法、请求地址、请求协议和请求头，如图 3-6 所示。单击“执行”按钮，即可执行所设置的请求，这里请求访问百度网站 www.baidu.com。

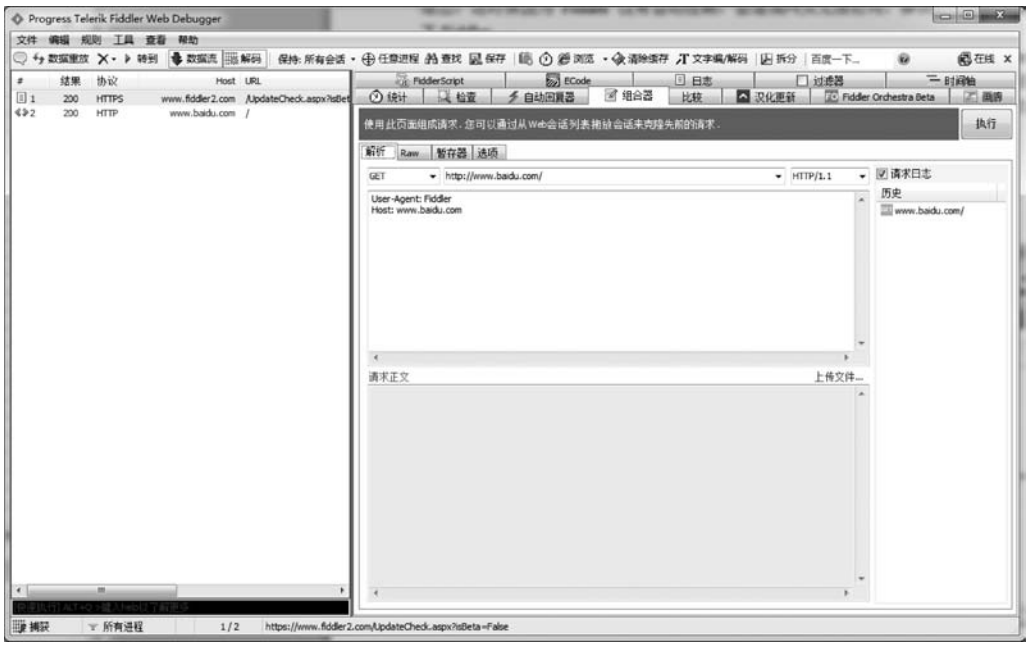

图 3-6 Fiddler“组合器”选项卡

图 3-6 左栏即是执行请求的结果，右击 www.baidu.com，在弹出的快捷菜单中选择“在新窗口查看”命令，在弹出的图 3-7 的“原始”选项卡中查看 raw 数据，raw 代表没有为了方便观看而格式化的原始数据。

```
GET http://www.baidu.com/ HTTP/1.1
User-Agent: Fiddler
Host: www.baidu.com
```

图 3-7 “原始”选项卡

在“响应”选项卡中可以查看请求的应答，如图 3-8 所示。

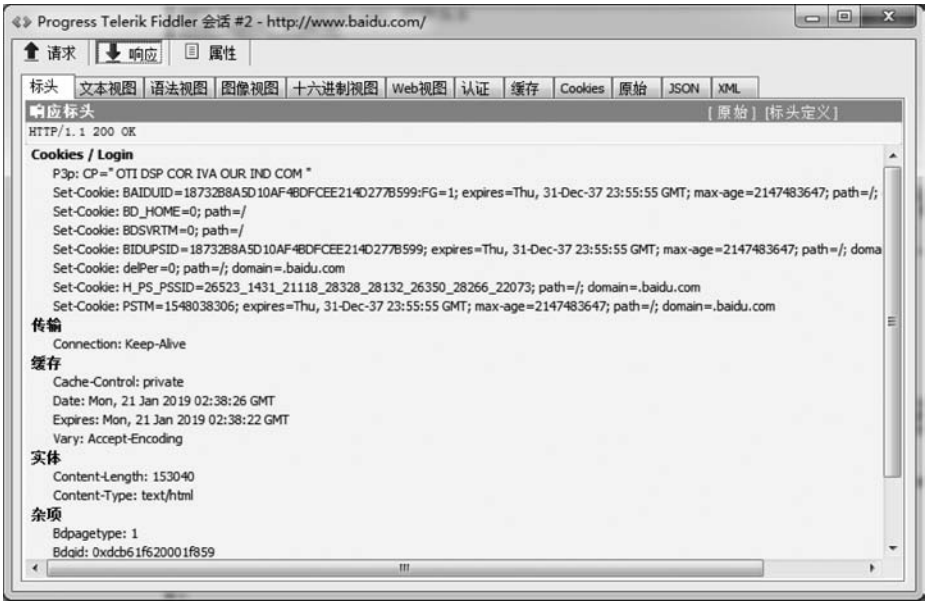

图 3-8 “响应”选项卡

3. 测试 POST 的请求和应答

POST 请求因为涉及需要上传的页面窗体数据，这里使用有道翻译来测试。如图 3-9 所示，登录 http://fanyi.youdao.com/，输入翻译数据“大家好”，单击“翻译”按钮后，被翻译数据“大家好”会以 POST 请求形式发给有道翻译服务器。图 3-10 所示为 Fiddler 抓取到的请求和响应信息。

图 3-9　有道翻译

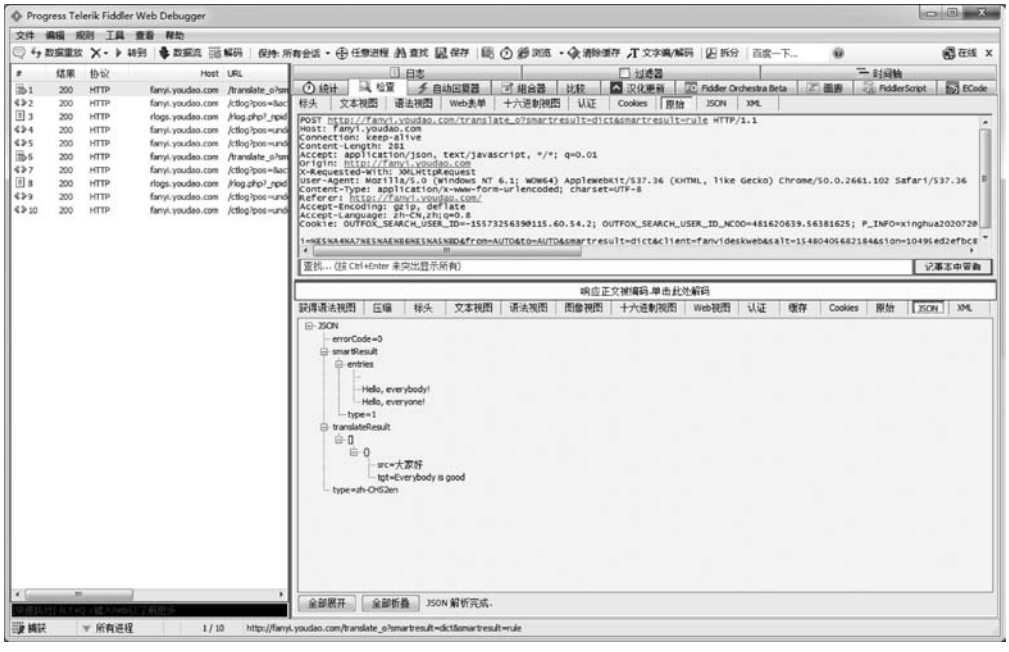

图 3-10　“检查”选项卡

可见其请求方法为 Post，请求地址为 http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule，请求协议为 HTTP/1.1。然后是请求头，请求头中有一个比较重要的设置如下：

```
Content-Type:application/x-www-form-urlencoded; charset=UTF-8
```

Content-Type 被指定为 application/x-www-form-urlencoded，这是最常见的 Post 提交数据的方式。浏览器的 form 表单如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数据。

此方式提交数据给服务器，提交的数据按照 key1=val1&key2=val2 的方式进行编码，key 和 val 都进行了 URL 转码。大部分服务端语言都对这种方式有很好的支持。本例如下：

```
i=%E5%A4%A7%E5%AE%B6%E5%A5%BD&from=AUTO&to=AUTO&smartresult=dict&client=
fanyideskweb&salt=15480405682184&sign=10495ed2efbc8fa6bffbca9420985af1&ts=
1548040568218&bv=4415abcea4c1a2e7f1af67e7e1b9742c&doctype=json&version=2.1&keyfrom=
fanyi.web&action=FY_BY_CLICKBUTTION&typoResult=false
```

很多时候，我们用 Ajax 提交数据时也是使用这种方式。例如 jQuery 的 Ajax，Content-Type 默认值都是“application/x-www-form-urlencoded;charset=utf-8”。

另外，Post 还有如下 3 种提交数据方式。

- Content-Type：application/json；charset=utf-8

表示会上传一个 json 文件，json 文件的格式是 utf-8，json 文件中保存传输给服务器的数据。

- Content-Type：multipart/form-data

使用表单上传文件时，必须让 form 的 enctyped 等于这个值。

- Content-Type：text/xml；charset=utf-8

表示会上传一个 XML 文件，采用 XML-RPC（XML Remote Procedure Call）协议。XML 作为编码方式的远程调用规范。

接着，在“检查”选项卡可得到 Post 请求的响应正文如下：

```
HTTP/1.1 200 OK
Server: nginx
Date: Mon, 21 Jan 2019 03:16:08 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Content-Encoding: gzip
```

可见 Content-Type：application/json；charset=utf-8，这说明在响应头结束后，会有一个 utf-8 编码的 JSON。选择“JSON”选项卡可以看到这个 json 的内容，如图 3-11 所示。

图 3-11　响应的 JSON 内容

从图 3-11 中的响应 JSON 内容可见，用户提交的被翻译数据“大家好”及翻译后的结果“Everybody is good”。在网络爬虫程序中解析 JSON 内容就可以获取翻译后的文本了。

3.2　Socket 编程

3.2.1　Socket 的概念

Socket 是网络编程的一个抽象概念。Socket 是套接字的英文名称，主要是用于网络通信编程。20 世纪 80 年代初，美国政府的高级研究工程机构（ARPA）给加利福尼亚大学 Berkeley 分校提供了资金，让他们在 UNIX 操作系统下实现 TCP/IP 协议。在这个项目中，研究人员为 TCP/IP 网络通信开发了一个应用程序接口（API）。这个 API 称为 Socket（套接字）。Socket 是 TCP/IP 网络最为通用的 API。任何网络通信都是通过 Socket 来完成的。

通常用一个 Socket 表示“打开了一个网络链接”，而打开一个 Socket 需要知道目标计算机的 IP 地址和端口号，再指定协议类型即可。

创建套接字需要使用套接字构造函数，套接字构造函数格式如下：

```
socket(family,type[,protocal])
```

使用给定的套接字家族、套接字类型、协议编号来创建套接字。

参数：

- family：套接字家族，可以使 AF_UNIX 或者 AF_INET、AF_INET6。
- type：套接字类型，可以根据是面向连接的还是非连接分为 SOCK_STREAM 或 SOCK_DGRAM。
- protocol：一般不填，默认为 0。

参数取值含义如表 3-3 所示。

表 3-3 参数含义

参 数	含 义
socket. AF_UNIX	只能够用于单一的 UNIX 系统进程间通信
socket. AF_INET	服务器之间网络通信
socket. AF_INET6	IPv6
socket. SOCK_STREAM	流式 Socket，针对 TCP
socket. SOCK_DGRAM	数据报式 Socket，针对 UDP
socket. SOCK_RAW	原始套接字。首先，普通的套接字无法处理 ICMP、IGMP 等网络报文，而 SOCK_RAW 可以；其次，SOCK_RAW 也可以处理特殊的 IPv4 报文；此外，利用原始套接字，可以通过 IP_HDRINCL 套接字选项由用户构造 IP 头
socket. SOCK_SEQPACKET	可靠的连续数据包服务

3.2.2 Socket 提供的函数方法

Socket 同时支持数据流 Socket 和数据报 Socket。

例如，创建 TCP Socket：

```
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

创建 UDP Socket：

```
s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
```

在 Python 中，Socket 模块中 Socket 对象提供的函数方法如表 3-4 所示。

表 3-4 Socket 对象函数方法

函 数	描 述
服务器端套接字	
s. bind(host,port)	绑定地址(host,port)到套接字，在 AF_INET 下以元组(host,port)的形式表示地址
s. listen(backlog)	开始 TCP 监听。backlog 指定在拒绝连接之前，可以连接的最大数量。该值至少为 1，大部分应用程序设为 5 就可以了
s. accept()	被动接受 TCP 客户端连接，(阻塞式)等待连接的到来
客户端套接字	
s. connect(address)	主动与 TCP 服务器连接。一般 address 的格式为元组(hostname,port)，如果连接出错，返回 socket. error 错误
s. connect_ex()	connect()函数的扩展版本，出错时返回出错码，而不是抛出异常
公共用途的套接字函数	
s. recv(bufsize,[,flag])	接收 TCP 数据，数据以字节串形式返回，bufsize 指定要接收的最大数据量。flag 提供有关消息的其他信息，通常可以忽略
s. send(data)	发送 TCP 数据，将 data 中的数据发送到连接的套接字。返回值是要发送的字节数量，该数量可能小于 data 的字节大小
s. sendall(data)	完整发送 TCP 数据，完整发送 TCP 数据。将 data 中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回 None，失败则抛出异常

续表

函　　数	描　　述
s. recvform(bufsize,[,flag])	接收 UDP 数据，与 recv()类似，但返回值是(data,address)。其中 data 是包含接收数据的字节串，address 是发送数据的套接字地址
s. sendto(data,address)	发送 UDP 数据，将数据发送到套接字，address 是形式为(ip,port)的元组，指定远程地址。返回值是发送的字节数
s. close()	关闭套接字
s. getpeername()	返回连接套接字的远程地址。返回值通常是元组(ipaddr,port)
s. getsockname()	返回套接字自己的地址。通常是一个元组(ipaddr,port)
s. settimeout(timeout)	设置套接字操作的超时时间，timeout 是一个浮点数，单位是秒。值为 None 表示没有超时时间。一般，超时时间应该在刚创建套接字时设置，因为它们可能用于连接的操作(如 connect())
s. gettimeout()	返回当前超时的值，单位是秒，如果没有设置超时，则返回 None
s. fileno()	返回套接字的文件描述符
s. makefile()	创建一个与该套接字相关联的文件

了解了 TCP/IP 协议的基本概念，以及 IP 地址、端口的概念和 Socket 后，就可以开始进行网络编程了。下面采用 TCP 协议类型来开发网络通信程序。

3.2.3　TCP 协议编程

TCP 可建立可靠连接，并且通信双方都可以以流的形式发送数据。图 3-12 所示为面向连接支持数据流 TCP 的时序图。

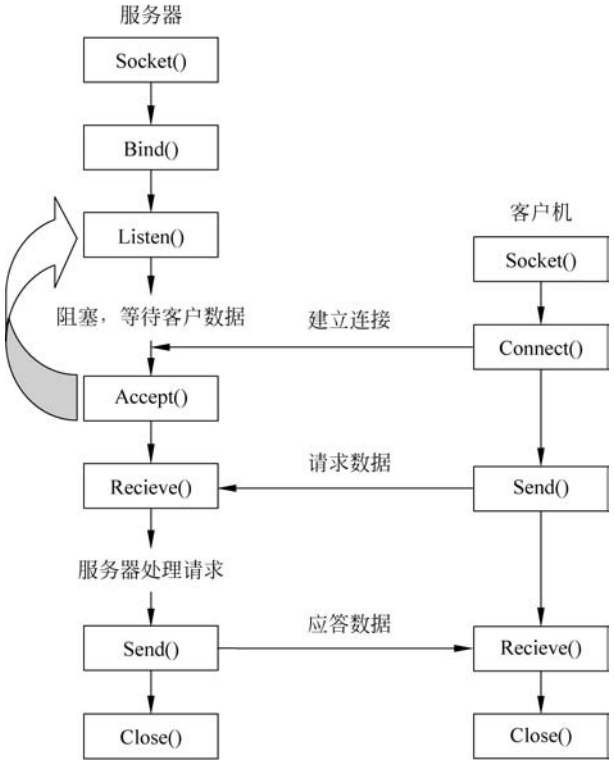

图 3-12　面向连接 TCP 的时序图

由图 3-3 可以看出，客户机(Client)与服务器(Server)的关系是不对称的。

对于 TCP C/S，服务器首先启动，然后在某一时刻启动客户机与服务器建立连接。服务器与客户机开始都必须调用 Socket()建立一个套接字 Socket，然后服务器调用 Bind()将套接字与一个本机指定端口绑定在一起，再调用 Listen()使套接字处于一种被动的准备接收状态，这时客户机建立套接字便可通过调用 Connect()和服务器建立连接。服务器就可以调用 Accept()来接收客户机连接。然后继续侦听指定端口，并发出阻塞，直到下一个请求出现，从而实现多个客户机连接。连接建立之后，客户机和服务器之间就可以通过连接发送和接收数据。最后，待数据传送结束，双方调用 Close()关闭套接字。

日常生活中大多数连接都是可靠的 TCP 连接。创建 TCP 连接时，主动发起连接的称为客户端，被动响应连接的称为服务器。

当在浏览器中访问当当购书网站时，用户自己的计算机就是客户端，浏览器会主动向当当网的服务器发起连接。如果一切顺利，当当的服务器接受了连接，一个 TCP 连接就建立起来了，后面的通信就是发送网页内容了。

【例 3-1】 使用 Socket 的 TCP 通信获取当当网首页的整个页面。

前面已经讲解了 HTTP 协议网络请求的原理，浏览器的主要功能是向服务器发出 HTTP 请求，获取响应报文，并且在浏览器窗口解析显示请求的网络资源。这里模拟浏览器向当当网发送一个 HTTP 的 Get 请求报文，请求获取当当网首页的 HTML 文件，服务器把包含该 HTML 文件的响应报文发送回本程序，从而获取整个网页的页面文件。

获取当当网首页的程序代码如下：

```
import socket                                                    #导入 socket 模块
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)            #创建一个 socket:
s.connect(('www.dangdang.com', 80))                              #建立与当当网站连接
#发送 HTTP 请求
s.send(b'GET / HTTP/1.1\r\nHost: www.dangdang.com\r\nConnection: close\r\n\r\n')
#接收数据:
buffer = []
while True:
    #每次最多接收 1KB:
    d = s.recv(1024)
    if d:
        buffer.append(d)
    else:
        break
#b"是一个空字节,join()是列表的函数,buffer 是一个字节串的列表,使用空字节把 buffer 这个字节列表连接在一起,成为一个新的字节串
data = b".join(buffer)
header, html = data.split(b'\r\n\r\n', 1)
print(header.decode('utf-8'))
#把接收的数据写入文件:
with open('当当.html', 'wb') as f:
    f.write(html)
```

代码中首先要创建一个基于 TCP 连接的 Socket：

```
import socket                                                    #导入 socket 模块
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)            #创建一个 socket:
s.connect(('www.sina.com.cn', 80))                               #建立与当当网站连接
```

创建 Socket 时，AF_INET 指定使用 IPv4 协议，如果要用更先进的 IPv6，就指定为 AF_INET6。SOCK_STREAM 指定使用面向流的 TCP 协议。这样，一个 Socket 对象就创建成功了，但是还没有建立连接。

客户端要主动发起 TCP 连接，必须知道服务器的 IP 地址和端口号。当当网站的 IP 地址可以用域名 www.dangdang.com 自动转换到 IP 地址，但是怎么知道当当服务器的端口号呢？

答案是作为服务器，提供什么样的服务，端口号就必须固定下来。由于用户想要访问网页，因此当当提供网页服务的服务器必须把端口号固定在 80 端口，因为 80 端口是 Web 服务的标准端口。其他服务都有对应的标准端口号，例如，SMTP 服务是 25 端口，FTP 服务是 21 端口等。端口号小于 1024 的是 Internet 标准服务的端口，端口号大于 1024 的，可以任意使用。

因此，连接当当服务器的代码如下：

```
s.connect(('www.dangdang.com ', 80))
```

注意参数是一个 tuple，包含地址和端口号。

建立 TCP 连接后，就可以向当当服务器发送请求，要求返回首页的内容：

```
#发送数据请求
s.send(b'GET / HTTP/1.1\r\nHost: www.dangdang.com \r\nConnection: close\r\n\r\n')
```

TCP 连接创建的是双向通道，双方都可以同时给对方发数据。但是谁先发谁后发，怎么协调，要根据具体的协议来决定。例如，HTTP 协议规定客户端必须先发请求给服务器，服务器收到后才发数据给客户端。

发送的文本格式必须符合 HTTP 标准，如果格式没问题，接下来就可以接收当当服务器返回的数据了：

```
#接收数据:
buffer = []
while True:
    d = s.recv(1024)          #每次最多接收 1KB
    if d:                     #是否为空数据
        buffer.append(d)      #字节串增加到列表中
    else:
        break                 #返回空数据,表示接收完毕,退出循环
data = b''.join(buffer)
```

接收数据时，调用 recv(max)方法，一次最多接收指定的字节数。因此，在一个 while 循环中反复接收，直到 recv()返回空数据，表示接收完毕，退出循环。

在 data=b".join(buffer)语句中,b"是一个空字节,join()是连接列表的函数,buffer 是一个字节串的列表,使用空字节把 buffer 这个字节列表连接在一起,成为一个新的字节串。这是 Python 3 的新功能,以前 join()函数只能连接字符串,现在可以连接字节串。

接收完数据后,调用 close()方法关闭 Socket。这样,一次完整的网络通信就结束了。

```
s.close()                    #关闭连接
```

接收到的数据包括 HTTP 头和网页本身,用户只需要把 HTTP 头和网页分离一下,把 HTTP 头打印出来,网页内容保存到文件:

```
header, html = data.split(b'\r\n\r\n', 1)   #以'\r\n\r\n'分割,且仅分割 1 次
print(header.decode('utf-8'))               #decode('utf-8')以 utf-8 编码将字节串转换成字符串
#把接收的数据写入文件:
with open('d:\\当当.html', 'wb') as f:      #以写方式打开文件'当当.html',即可以写入信息
    f.write(html)
```

程序运行后,在 D 盘上可以看到“当当.html”文件,也就是用户把当当的首页爬到本机了。现在,只需要在浏览器中打开这个“当当.html”文件,就可以看到当当的首页了。由于许多网站(如新浪网站)现已改成 HTTPS 安全传输协议,HTTPS 在 HTTP 的基础上加入了 SSL 协议,SSL 依靠证书来验证服务器的身份,并为浏览器和服务器之间的通信加密。读者可以换成其他网站(如当当网 www.dangdang.com),这样仍可以采用 HTTP 传输协议测试本例。

而 HTTPS 传输协议需要使用 SSL 模块,所以采用 HTTPS 协议访问新浪网站代码如下:

```
import socket                                  #导入 socket 模块
import ssl
#s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)     #创建一个 socket
s = ssl.wrap_socket(socket.socket())
s.connect(('www.sina.com.cn', 443))            #建立与新浪网站连接
#发送数据请求
s.send(b'GET / HTTP/1.1\r\nHost:www.sina.com.cn\r\nConnection: close\r\n\r\n')
#接收数据:
buffer = []
while True:
    d = s.recv(1024)                           #每次最多接收服务器端 1KB 数据
    if d:                                      #是否为空数据
       buffer.append(d)                        #字节串增加到列表中
    else:
        break                                  #返回空数据,表示接收完毕,退出循环
data = b".join(buffer)
header, html = data.split(b'\r\n\r\n', 1)
print(header.decode('utf-8'))
#把接收的数据写入文件:
with open('d:\\sina.html', 'wb') as f:
   f.write(html)
```

运行结果如下：

```
HTTP/1.1 200 OK
Server: edge - esnssl - 1.12.1 - 13
Date: Wed, 12 Dec 2018 14:37:14 GMT
Content - Type: text/html
Content - Length: 572032
Connection: close
Last - Modified: Wed, 12 Dec 2018 14:36:02 GMT
Vary: Accept - Encoding
X - Powered - By: shci_v1.03
Expires: Wed, 12 Dec 2018 14:38:06 GMT
Cache - Control: max - age = 60
Age: 10
Via: https/1.1 cnc.beixian.ha2ts4.205 (ApacheTrafficServer/6.2.1 [cMsSfW]), https/1.1 cnc.
zhengzhou.ha2ts4.196 (ApacheTrafficServer/6.2.1 [cHs f ])
X - Via - Edge: 1544625434199db313c73f4fb9e3d2b04c19e
X - Cache: HIT.196
X - Via - CDN: f = edge,s = cnc.zhengzhou.edssl.211.nb.sinaedge.com,c = 115.60.49.219;f = edge,
s = cnc.zhengzhou.ha2ts4.196.nb.sinaedge.com,c = 61.158.251.211;f = Edge,s = cnc.zhengzhou.
ha2ts4.196,c = 61.158.251.196
```

HTTP 响应的头信息中 HTTP/1.1 200 OK 表示访问成功，而不再出现 HTTP/1.1 302 Moved Temporarily 错误。

通过上面的例子，可以掌握采用底层 Socket 编程实现浏览网页的过程，熟悉 HTTP 通信过程。在实际的爬虫开发中，使用 Python 网页访问的标准库 urllib、第三方库 requests 等浏览网页更加容易，后面爬虫的开发主要使用 urllib、requests 库来实现浏览网页、抓取网页。