# 第1章 爬虫基础

## 1.1 HTTP基本原理

了解在浏览器输入URL到获取网页内容之间发生了什么？

### 1.1.1 URI和URL

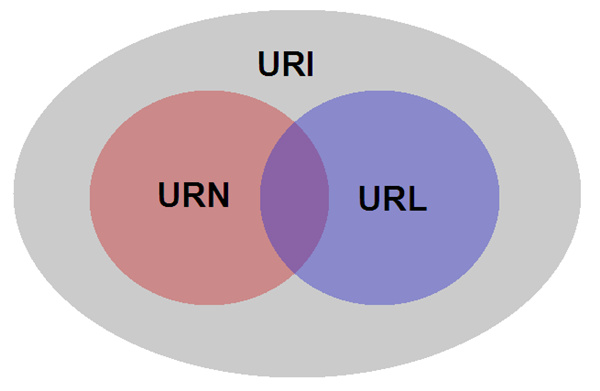
URL (Uniform Resource Identifier)，即统一资源标志符；

URL (Universal Resource Locator)，即统一资源定位符；

URN (Uniersal Resource Name)，即统一资源名称；

举例来说，https://github.com/favicon.ico是GitHub的网站图标链接，它是一个URL，也是一个URI。即有一个图标资源，我们用URL/URI来唯一指定了它的访问方式，这其中包括了访问协议https、访问路径（即/根目录）和资源名称favicon.ico。通过这样一个链接，我们便可以从互联网找到这个资源，这就是URL/URI。

**URL是URI的子集，也就是说每个URL都是URI，但不是每个URI都是URL**。另外，URN只命名资源而不指定如何定位资源，比如urn:isbn:0451450523指定了一本书的ISBN，可以唯一标识这本书，但是没有指定到哪里定位这本书，这就是URN。URL,URN和URI的三者关系可以标识为：



注意：在所有的互联网中，URN用的非常少，几乎所有的URI都是URL，一般的网页链接我们既可以称为URL，也可以称为URI。

### 1.1.2 超文本

hypertext ==> 超文本

超文本标记语言：HTML

### 1.2.3 HTTP和HTTPS

是一种协议类型。

HTTP全称是Hyper Text Transfer Protocol，中文名叫**超文本传输协议**。

HTTP协议是用于**从网络传输超文本数据到本地浏览器**的传送协议，它能保证高效而准确的传送超文本文档。

HTTPS的全称是Hyper Text Transfer Protocol over Secure Socket Layer，是以安全为目标的HTTP通道，简单讲是**HTTP的安全版，即HTTP下加入SSL层，简称HTTPS**。

HTTPS的安全基础是SSL，因此通过它传输的内容都是经过SSL加密的，它的主要作用可以分为以下：

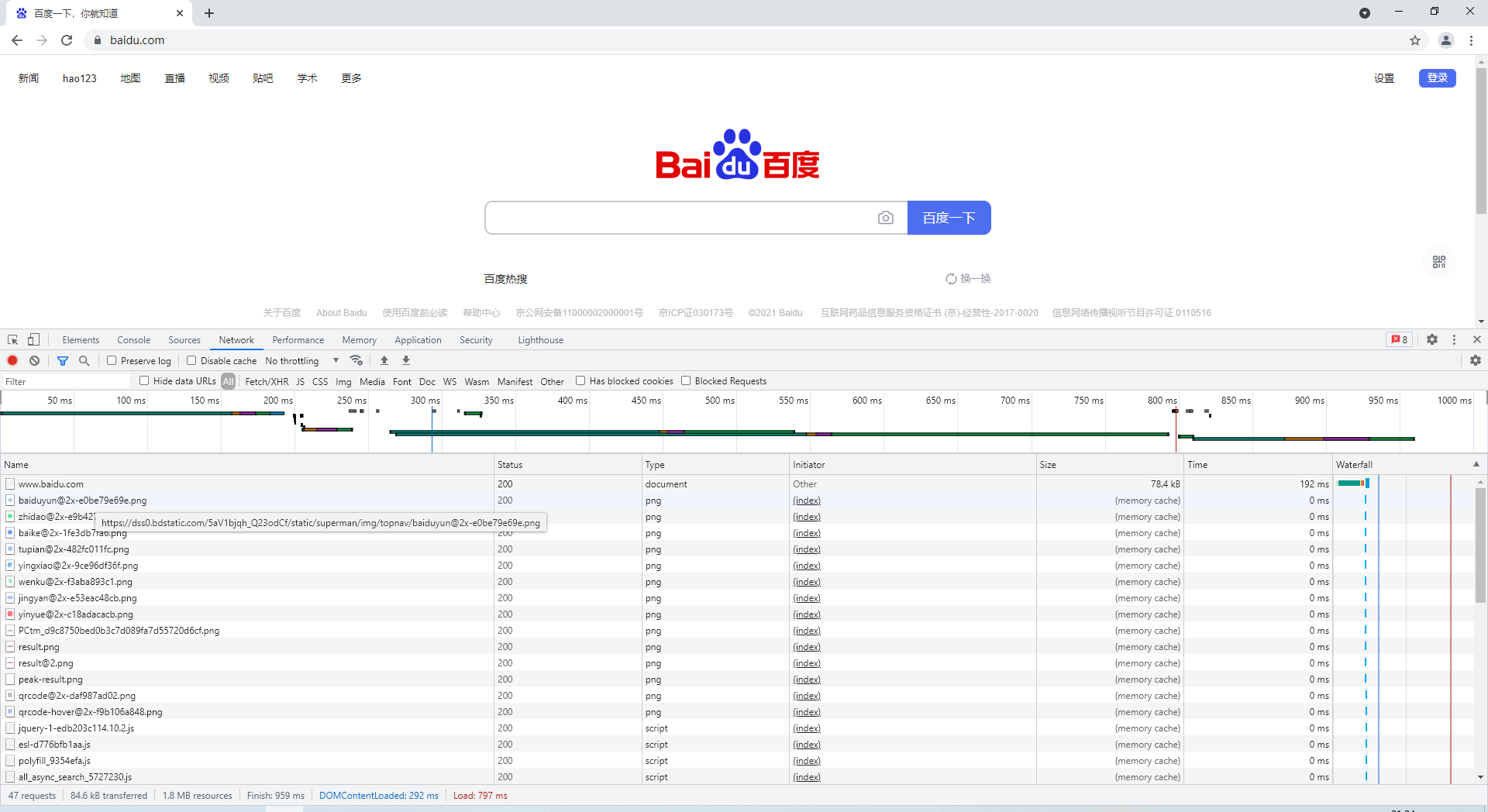
（1）建立一个信息安全通道来保证数据传输的安全；

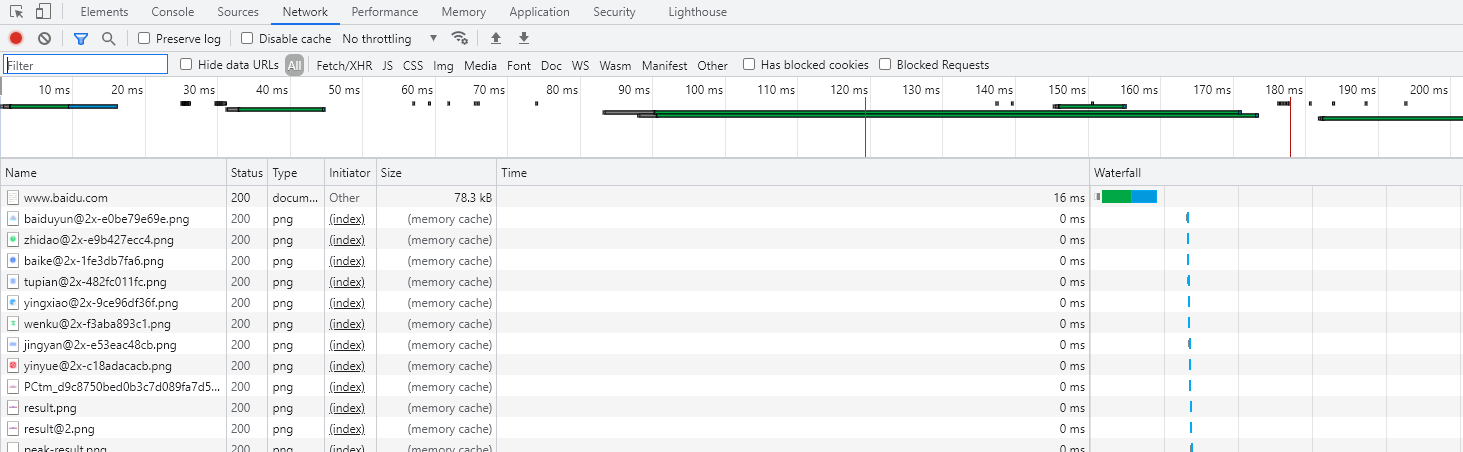
（2）确认网站的真实性，凡事使用HTTPS的网站，都是可以通过点击浏览器地址栏的锁头标志来查看网站认证后的真实信息，也可以通过CA机构颁发的安全签章来查询。

### 1.2.4 HTTP的请求过程

我们在浏览器中输入一个地址栏，回车之后就会在浏览器中观察到页面内容。

实际上，这个过程是**浏览器向网站所在的服务器发送了一个请求，网站服务器接收到这个请求后进行处理和解析，然后返回对应的响应，接着传回给浏览器**。响应里包含了页面的源代码等内容，浏览器再进行解析，便将网页呈现出来了。





F12之后，点击Network一栏中，各字段如下：

Name：请求的名称，一般会将URL 的最后一部分内容当作名称。

Status：请求的状态码，这里显示为200，代表响应是正常的。

Type：请求的文档类型。这里为 document，代表我们这次请求的是一个 HTML 文档，内容就是一些 HTML 代码。

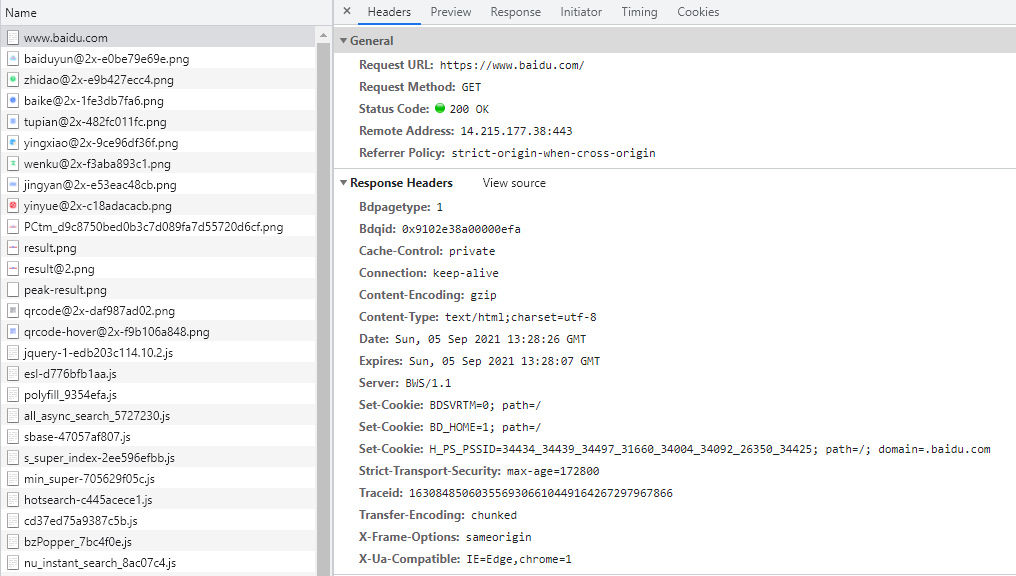
Initiator：请求源。用来标记请求是由哪个对象或进程发起的。

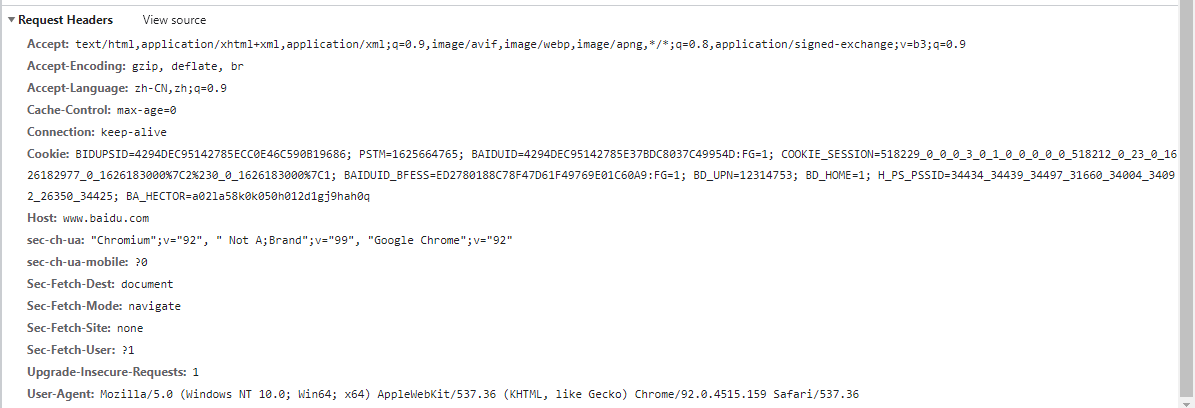
Size：从服务器下载的文件和请求的资源大小。如果是从缓存中取得的资源，则该列会显示 from cache。

Time：发起请求到获取响应所用的总时间。

Waterfall：网络请求的可视化瀑布流。

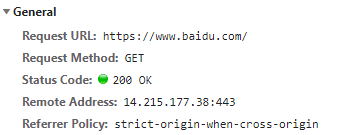
点击这个条目可以看到详情





首先是 General 部分，Request URL 为请求的 URL，Request Method 为请求的方法，Status Code 为响应状态码，Remote Address 为远程服务器的地址和端口，Referrer Policy 为 Referrer 判别策略。

General部分：



Request URL是请求的URL；

Request Method是请求的方法；

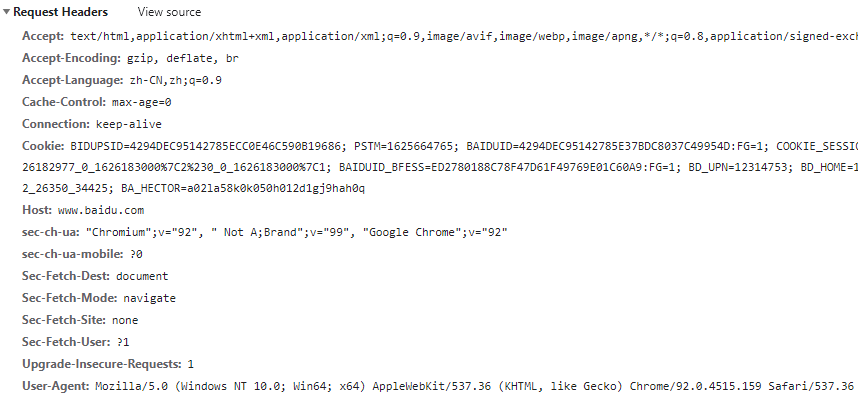
Status Code是响应状态码；

Remote Address是远程服务的地址和端口；

Referrer Policy为Referrer判别策略；

再继续往下看，可以看到，有 **Response Headers** 和 **Request Headers**，这分别代表响应头和请求头。请求头里带有许多请求信息，例如浏览器标识、Cookies、Host 等信息，这是请求的一部分，服务器会根据请求头内的信息判断请求是否合法，进而作出对应的响应。图中看到的 Response Headers 就是响应的一部分，例如其中包含了服务器的类型、文档类型、日期等信息，浏览器接受到响应后，会解析响应内容，进而呈现网页内容。

Request Header：请求头



Response Headers：响应头



### 1.2.5 请求

请求，由客户端向服务端发出，可以分为4个部分：请求方法（Request Method）、请求的网址（Request URL）、请求头（Request Header）和请求体（Request Body）。

（1）请求方法

常见的方法是：GET和POST。

在浏览器中直接输入 URL 并回车，这便发起了一个 GET 请求，请求的参数会直接包含到 URL里。例如，在百度中搜索Python，这就是一个GET 请求，链接为 <https://www.baidu.com/s?wd=Python>，其中 URL 中包含了请求的参数信息，这里参数 wd 表示要搜寻的关键字。POST 请求大多在表单提交时发起。比如，对于一个登录表单，输入用户名和密码后，点击 “登录” 按钮，这通常会发起一个 POST 请求，其数据通常以表单的形式传输，而不会体现在 URL 中。

GET和POST的区别如下：

* GET请求的参数包含在URL里面，数据可以在URL中看到，而POST请求的URL不会包含这些数据，数据都是通过表单形式传输的，会包含在请求体中；
* GET请求提交的数据最多只有1024字节，而POST方式没有限制；

因此，在登陆时，需要提交用户名和密码，其中包含了敏感信息，使用GET请求，密码会暴露在URL中，造成密码泄露，因此使用POST请求合适。上传文件时，由于内容比较大，也会采用POST的方式。

除此之外，还有其他的请求方式：



（2）请求的网址

请求的网址，即统一资源定位符URL，它可以唯一确定我们请求的资源。

（3）请求头

请求头用来说明服务器要使用的附加信息，比较重要的是Cookie，Referer，User-Agent等。

Accept：请求报头域，用于指定客户端可接受哪些类型的信息。

Accept-Language：指定客户端可接受的语言类型。

Accept-Encoding：指定客户端可接受的内容编码。

Host：用于指定请求资源的主机IP和端口号，其内容为请求URL的原始服务器或网关的位置。从HTTP1.1版本开始，请求必须包含此内容。

Cookie：也常用复数形式 Cookies，这是网站为了辨别用户进行会话跟踪而存储在用户本地的数据。它的主要功能是**维持当前访问会话**。例如，我们输入用户名和密码成功登陆某个网站后，服务器会用会话保持登陆状态信息，后面我们每次刷新或请求该网站的其他页面时，会发现都是登陆状态，这就是Cookie的功劳。Cookie里有信息标识了我们所对应的服务器的会话，每次浏览器在请求该站点的页面上时，都会在请求头中加上Cookie并将其发送给服务器，服务器通过Cookie识别出我们自己，并且检查出当前状态是登陆状态，所以返回结果就是登陆之后才能看到的网页内容。

Referer：此内容用来标记这个请求是从哪个页面发过来的，服务器可以拿到这一信息并做相应的处理，如做来源统计、防盗链处理。

User-Agent：简称UA，它是一个特殊的字符串头，可以使服务器识别客户端使用的操作系统及版本、浏览器及版本等信息。在做爬虫时加上此信息，可以伪装成浏览器。

Content-Type：也叫互联网媒体类型或者MIME类型，在HTTP协议消息头中，它用来表示具体请求中的媒体类型信息。

因此，请求头是请求的重要组成部分，在写爬虫时，大部分情况下都需要设定请求头。

（4）请求体

请求体一般承载的内容是POST请求中表单数据，而对于GET请求，请求体则为空。

Content-Type和POST提交数据方式的关系：

|  |  |
| --- | --- |
| Content-Type | 提交数据的方式 |
| application/x-www.form-urlencoded | 表单数据 |
| multipart/form-data | 表单文件上传 |
| application/json | 序列号JSON数据 |
| text/html | XML数据 |

在爬虫中，如果要构造 POST 请求，需要使用正确的 Content-Type，并了解各种请求库的各个参数设置时使用的是哪种 Content-Type，不然可能会导致 POST 提交后无法正常响应。

### 1.2.6 响应

响应，由服务端返回给客户端，可以分为三个部分：响应状态码（Response Status Code），响应头（Response Headers）和响应体（Response Body）。

（1）响应状态码

响应状态码表示服务器的状响应态，如200代表服务器正常响应，404 代表页面未找到，500 代表服务器内部发生错误。在爬虫中，我们可以根据状态码来判断服务器响应状态，如状态码为 200，则证明成功返回数据，再进行进一步的处理，否则直接忽略。在爬虫中，我们可以根据状态码来判断服务器响应状态。

常见的错误码及原因：

|  |  |  |
| --- | --- | --- |
| 状态码 | 说明 | 详情 |
| 100 | 继续 | 请求者应当继续提出请求。服务器已收到请求的一部分，正在等待其余部分 |
| 101 | 切换协议 | 请求者已要求服务器切换协议，服务器已确认并准备切换 |
| 200 | 成功 | 服务器已成功处理了请求 |
| 201 | 已创建 | 请求成功并且服务器创建了新的资源 |
| 202 | 已接收 | 服务器已接受请求，但尚未处理 |
| 203 | 非授权信息 | 服务器已成功处理了请求，但返回的信息可能来自另一个源 |
| 204 | 无内容 | 服务器成功处理了请求，但没有返回任何内容 |
| 205 | 重置内容 | 服务器成功处理了请求，内容被重置 |
| 206 | 部分内容 | 服务器成功处理了部分请求 |
| 300 | 多种选择 | 针对请求，服务器可执行多种操作 |
| 301 | 永久移动 | 请求的网页已永久移动到新位置，即永久重定向 |
| 302 | 临时移动 | 请求的网页暂时跳转到其他页面，即暂时重定向 |
| 303 | 查看其他位置 | 如果原来的请求是 POST，重定向目标文档应该通过 GET 提取 |
| 304 | 未修改 | 此次请求返回的网页未修改，继续使用上次的资源 |
| 305 | 使用代理 | 请求者应该使用代理访问该网页 |
| 307 | 临时重定向 | 请求的资源临时从其他位置响应 |
| 400 | 错误请求 | 服务器无法解析该请求 |
| 401 | 未授权 | 请求没有进行身份验证或验证未通过 |
| 403 | 禁止访问 | 服务器拒绝此请求 |
| 404 | 未找到 | 服务器找不到请求的网页 |
| 405 | 方法禁用 | 服务器禁用了请求中指定的方法 |
| 406 | 不接受 | 无法使用请求的内容响应请求的网页 |
| 407 | 需要代理授权 | 请求者需要使用代理授权 |
| 408 | 请求超时 | 服务器请求超时 |
| 409 | 冲突 | 服务器在完成请求时发生冲突 |
| 410 | 已删除 | 请求的资源已永久删除 |
| 411 | 需要有效长度 | 服务器不接受不含有效内容长度标头字段的请求 |
| 412 | 未满足前提条件 | 服务器未满足请求者在请求中设置的其中一个前提条件 |
| 413 | 请求实体过大 | 请求实体过大，超出服务器的处理能力 |
| 414 | 请求 URI 过长 | 请求网址过长，服务器无法处理 |
| 415 | 不支持类型 | 请求格式不被请求页面支持 |
| 416 | 请求范围不符 | 页面无法提供请求的范围 |
| 417 | 未满足期望值 | 服务器未满足期望请求标头字段的要求 |
| 500 | 服务器内部错误 | 服务器遇到错误，无法完成请求 |
| 501 | 未实现 | 服务器不具备完成请求的功能 |
| 502 | 错误网关 | 服务器作为网关或代理，从上游服务器收到无效响应 |
| 503 | 服务不可用 | 服务器目前无法使用 |
| 504 | 网关超时 | 服务器作为网关或代理，但是没有及时从上游服务器收到请求 |
| 505 | HTTP 版本不支持 | 服务器不支持请求中所用的 HTTP 协议版本 |

（2）响应头

响应头包含了服务器对请求的应答信息，如Content-Type，Server，Set-Cookie等。常用如下：

Date：标识响应产生的时间

Last-Modified：指定资源的最后修改时间

Content-Encoding：指定响应内容的编码

Server：包含服务器的信息，比如名称，版本号等

Content-Type：文档类型，指的是返回的数据类型是什么，如 text/html 代表返回 HTML 文档，application/x-javascript 则代表返回 JavaScript 文件，image/jpeg 则代表返回图片。

Set-Cookie：设置Cookies。响应头中的Set-Cookie告诉浏览器需要将此内容放在Cookies中，下次请求携带Cookies请求。

Expires：指定响应的过期时间，可以使代理服务器或浏览器将加载的内容更新到缓存中。如果再次访问时，就可以直接从缓存中加载，降低浏览器负载，缩短加载时间。

（3）响应体

响应的正文数据都在响应体中，比如请求网页时，它的响应体就是网页的HTML代码；请求一张图片时，它的响应体就是图片的二进制数据。做爬虫时，要解析的内容就是响应体。

在浏览器开发者工具中点击 Preview，就可以看到网页的源代码，也就是响应体的内容，它是解析的目标。

在做爬虫时，我们主要通过响应体得到网页的源代码、JSON 数据等，然后从中做相应内容的提取。

## 1.2 网页基础

### 1.2.1 网页的组成

HTML相当于骨架，JS相当于肌肉，CSS相当于皮肤，三者结合起来才是一个完整的网页。

（1）HTML

Hyper Text Markup Language 即超文本标记语言。

（2）CSS

Cascading Style Sheets 即层叠样式表。

（3）Javascript

一种弱类型的脚本语言。

### 1.2.2 网页的结构

### 1.2.3 节点树及节点间的关系

### 1.2.4 选择器

## 1.3 爬虫的基本原理

### 1.3.1 爬虫概述

简单而言，爬虫就是获取网页并提取和保存信息的自动化程序。

（1）获取网页

爬虫首先要做的就是获取网页，这里就是获取网页的源代码。

前面讲到请求和响应的概念，向网站的服务器发送了一个请求，返回的响应体便是网页源代码。所以最关键的部分就是构造一个请求并发送给服务器，然后接收到响应体并将其解析出来。

（2）提取信息

在获取网页源代码之后，接下来就是分析网页源代码，从中提取我们想要的数据。有如下方法：

1）使用正则表达式，这个一个万能的方法，但是在构造正则表达式时比较复杂且容易出错；

2）使用CSS或XPath选择器，这是根据网页结构的规则来提取的，如Beautiful Soup、Pyquery、lxml等；

提取信息时爬虫中非常重要的部分，他可以使杂乱的数据变得条例清晰，以便我们后续处理和分析数据。

（3）保存数据

保存到TXT/CSV/JSON格式的文本，也可以保存到数据库中，还有远程服务器中。

（4）自动化程序

当量特别大或者快速获取大量数据时，需要自动化爬虫程序，它可以在抓取过程中进行各种异常处理，错误重试等操作，确保爬取持续高效的运行。

### 1.3.2 能抓怎样的数据

（1）常见的HTML代码；

（2）JSON字符串（API接口大多采用这种方式）；

（3）二进制数据（图片，视频，音频等）；

（4）其他Css，JS和配置文件；

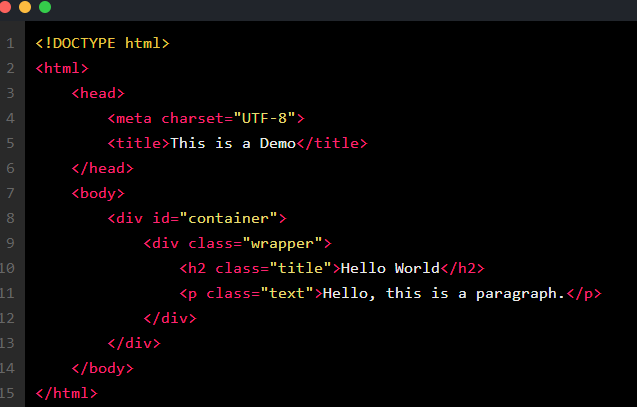
### 1.3.3 JavaScript渲染页面

有时候，我们在用urllib或requests抓取网页时，得到的源代码实际和浏览器中看到的不一样。网页不但只是简单的HTML，更多的采用JS来渲染这些网页，使用普通的urllib或requests模块无法解决这些问题，因此，需要分析其后台Ajax接口，也可以使用Selenium/Splash这样的库来模拟JS渲染的页面。

## 1.4 会话和Cookies

在浏览网站的过程中，我们经常会遇到需要登录的情况，有些页面只有登录之后才可以访问，而且登录之后可以连续访问很多次网站，但是有时候过一段时间就需要重新登录。还有一些网站，在打开浏览器时就自动登录了，而且很长时间都不会失效，这种情况又是为什么？其实这里面涉及会话和 Cookies 的相关知识1.4.1 静态网页和动态网页

### 1.4.1 静态网页和动态网页



这是最基本的 HTML 代码，我们将其保存为一个.html 文件，然后把它放在某台具有固定公网 IP 的主机上，主机上装上 Apache 或 Nginx 等服务器，这样这台主机就可以作为服务器了，其他人便可以通过访问服务器看到这个页面，这就搭建了一个最简单的网站。

这种网页的内容是 HTML 代码编写的，文字、图片等内容均通过写好的 HTML 代码来指定，这种页面叫作**静态网页**。它加载速度快，编写简单，但是存在很大的缺陷，如可维护性差，不能根据 URL 灵活多变地显示内容等。例如，我们想要给这个网页的 URL 传入一个 name 参数，让其在网页中显示出来，是无法做到的。

因此，动态网页应运而生，它可以动态解析 URL 中参数的变化，关联数据库并动态呈现不同的页面内容，非常灵活多变。我们现在遇到的大多数网站都是动态网站，它们不再是一个简单的 HTML，而是可能由 JSP、PHP、Python 等语言编写的，其功能比静态网页强大和丰富太多了。

此外，动态网站还可以实现用户登录和注册的功能。再回到开头提到的问题，很多页面是需要登录之后才可以查看的。按照一般的逻辑来说，输入用户名和密码登录之后，肯定是拿到了一种类似凭证的东西，有了它，我们才能保持登录状态，才能访问登录之后才能看到的页面。

那么，这种神秘的凭证到底是什么呢？其实它就是会话和 Cookies 共同产生的结果，下面我们来一探究竟。

### 1.4.2 无状态HTTP

HTTP的一大特点：无状态指的是**HTTP协议对事务处理是没有记忆能力的，也就是说服务器不知道客户端是什么状态**。

当我们向服务器发送请求时后，服务器会解析此请求，然后返回对应的响应，服务器负责完成这个过程，而这个过程是完全独立的，服务器不会记录前后状态的变化，也就是缺少状态记录。这意味着后续如果需要处理前面的信息，则必须重传，这导致需要额外传递一些前面重复的请求，才能获取后续响应，然而这种效果显然不是我们后续想要的。为了保持登陆前后状态，我们需要避免资源的浪费。

这时，用于保持HTTP连接状态的技术出现了，即Session和Cookie。**Session在服务端，也就是网站的服务器，用于保存用户的会话信息；Cookie在客户端，可以理解成浏览器**；有了Cookie，浏览器在下次访问网页时会自动附带上它发送给服务器，服务器通过辨识Cookie并鉴定出是哪个用户，然后再判断用户是否是登陆状态，然后返回对应的响应。可以理解Cookie里面保存了登陆的凭证，有了它，只需要在下次请求携带Cookie发送请求而不必重新输入用户名、密码等信息重新登陆了。

因此在爬虫中，一般将登陆成功后获取的Cookie放在请求头里面直接请求，而不必模拟重新登陆。

（1）会话（Session）

会话，其本来的含义是指有始有终的一系列动作/消息。如打电话时，从拿起电话拨号到挂断电话这中间的一系列过程可以称为一个会话。

**而在Web中，会话对象用来存储特定用户会话所需的属性及配置信息**。这样，当用户在应用程序的Web页之间跳转时，存储在会话对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。当用户请求来自应用程序的Web页时，如果该用户还没有会话，则Web程序将自动创建一个会话对象。当会话过期或被放弃时，服务器将终止该会话。

（2）Cookies

**Cookies指的是某些网站为了辨别用户身份、进行会话跟踪而存储在用户本地终端的数据**。

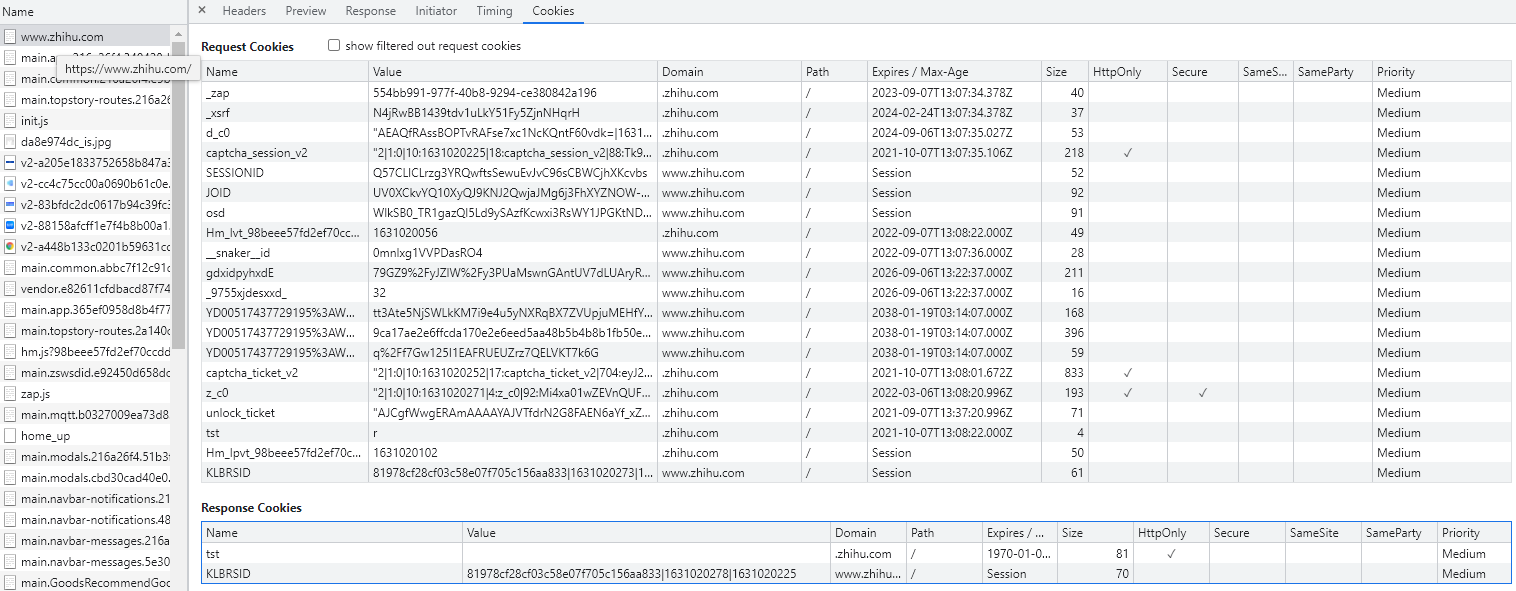
会话维持：

当客户端第一次请求服务器时，服务器会返回一个响应头中带有Set-Cookie字段的响应给客户端；用来标记是哪一个用户，客户端浏览器会把Cookies保存起来。当浏览器下一次请求该网站时，浏览器会把此Cookies放到请求头中一起提交给服务器，Cookies携带了会话ID信息，服务器检查该Cookies即可找到对应的会话是什么，然后再判断会话来以此辨认用户状态。

在成功登陆某个网站时，服务器会告诉客户端设置哪些Cookies信息，在后续访问页面时客户端会把Cookies发送给服务器，服务器再找到对应的会话并加以判断。如果会话中的某些设置登陆状态的变量是有效的，那么证明用户处于登陆状态，此时返回登陆之后才可以查看的网页内容，浏览器再进行解析便可以看到了。

反之，如果传给服务器的Cookies是无效的，或者会话已经过期了，我们将不能继续访问页面，此时可能会收到错误的响应或者跳转到登陆页面重新登陆。

属性结构：



有如下几个字段：

Name：该Cookies的名称。一旦创建，不可更改。

Value：该Cookies的值。如果值时Unicode字符，需要字符编码，如果是二进制数据，需要BASE64编码

Domain：可以访问该Cookie的域名。例如，如果设置为.zhihu.com，则所有以 zhihu.com，结尾的域名都可以访问该 Cookie。

Max-Age：该 Cookie 失效的时间，单位为秒，也常和 Expires 一起使用，通过它可以计算出其有效时间。Max Age 如果为正数，则该 Cookie 在 Max Age 秒之后失效。如果为负数，则关闭浏览器时 Cookie 即失效，浏览器也不会以任何形式保存该 Cookie。

Path：该Cookie的使用路径。如果设置为 /path/，则只有路径为 /path/ 的页面可以访问该 Cookie。如果设置为 /，则本域名下的所有页面都可以访问该 Cookie。

Size：该Cookie的大小。

Http字段：Cookie的HttpOnly属性。若此属性为 true，则只有在 HTTP 头中会带有此 Cookie 的信息，而不能通过 document.cookie 来访问此 Cookie。

Secure：该HttpOnly是否被安全协议传输。安全协议有 HTTPS 和 SSL 等，在网络上传输数据之前先将数据加密。默认为 false。

会话Cookie和持久Cookie：

从表面意思来说，会话 Cookie 就是把 Cookie 放在浏览器内存里，浏览器在关闭之后该 Cookie 即失效；持久 Cookie 则会保存到客户端的硬盘中，下次还可以继续使用，用于长久保持用户登录状态。

其实严格来说，没有会话 Cookie 和持久 Cookie 之分，只是由 Cookie 的 Max Age 或 Expires 字段决定了过期的时间。

因此，一些持久化登录的网站其实就是把 Cookie 的有效时间和会话有效期设置得比较长，下次我们再访问页面时仍然携带之前的 Cookie，就可以直接保持登录状态。

### 1.4.3 常见误区

在谈论会话机制的时候，常常听到这样一种误解 “只要关闭浏览器，会话就消失了”，这种理解是错误的。可以想象一下会员卡的例子，除非顾客主动对店家提出销卡，否则店家绝对不会轻易删除顾客的资料。对会话来说，也是一样，除非程序通知服务器删除一个会话，否则服务器会一直保留。比如，程序一般都是在我们做注销操作时才去删除会话。

但是当我们关闭浏览器时，浏览器不会主动在关闭之前通知服务器它将要关闭，所以服务器根本不会有机会知道浏览器已经关闭。之所以会有这种错觉，是因为大部分会话机制都使用会话 Cookie 来保存会话 ID 信息，而关闭浏览器后 Cookies 就消失了，再次连接服务器时，也就无法找到原来的会话了。如果服务器设置的 Cookies 保存到硬盘上，或者使用某种手段改写浏览器发出的 HTTP 请求头，把原来的 Cookies 发送给服务器，则再次打开浏览器，仍然能够找到原来的会话 ID，依旧还是可以保持登录状态的。

而且恰恰是由于关闭浏览器不会导致会话被删除，这就需要服务器为会话设置一个失效时间，当距离客户端上一次使用会话的时间超过这个失效时间时，服务器就可以认为客户端已经停止了活动，才会把会话删除以节省存储空间。

## 1.5 代理的基本原理

我们在做爬虫的过程中经常会遇到这样的情况，最初爬虫正常运行，正常抓取数据，一切看起来都是那么美好，然而一杯茶的功夫可能就会出现错误，比如 403 Forbidden，这时候打开网页一看，可能会看到 “您的 IP 访问频率太高” 这样的提示。出现这种现象的原因是网站采取了一些反爬虫措施。比如，服务器会检测某个 IP 在单位时间内的请求次数，如果超过了这个阈值，就会直接拒绝服务，返回一些错误信息，这种情况可以称为封 IP。

既然服务器检测的是某个 IP 单位时间的请求次数，那么借助某种方式来伪装我们的 IP，让服务器识别不出是由我们本机发起的请求，不就可以成功防止封 IP 了吗？

一种有效的方式就是使用代理，后面会详细说明代理的用法。在这之前，需要先了解下代理的基本原理，它是怎样实现 IP 伪装的呢？

### 1.5.1 基本原理

代理实际上指的就是代理服务器，英文叫作 proxy server，它的功能是**代理网络用户去取得网络信息**。形象地说，它是网络信息的中转站。在我们正常请求一个网站时，是发送了请求给 Web 服务器，Web 服务器把响应传回给我们。如果设置了代理服务器，实际上就是在本机和服务器之间搭建了一个桥，此时本机不是直接向 Web 服务器发起请求，而是向代理服务器发出请求，请求会发送给代理服务器，然后由代理服务器再发送给 Web 服务器，接着由代理服务器再把 Web 服务器返回的响应转发给本机。这样我们同样可以正常访问网页，但这个过程中 Web 服务器识别出的真实 IP 就不再是我们本机的 IP 了，就成功实现了 IP 伪装，这就是代理的基本原理。

### 1.5.2 代理的作用

* 突破自身IP的限制，访问一些平时不能访问的站点；
* 访问一些单位或团体内部资源：比如使用教育网内地址段免费代理服务器，就可以用于对教育网开放的各类 FTP 下载上传，以及各类资料查询共享等服务；
* 提高访问速度：通常代理服务器都设置一个较大的硬盘缓冲区，当有外界的信息通过时，同时也将其保存到缓冲区中，当其他用户再访问相同的信息时，则直接由缓冲区中取出信息，传给用户，以提高访问速度；
* 隐藏真实IP: 上网者也可以通过这种方法隐藏自己的 IP，免受攻击。对于爬虫来说，我们用代理就是为了隐藏自身 IP，防止自身的 IP 被封锁；

### 1.5.3 爬虫代理

对于爬虫来说，由于爬虫爬取速度过快，在爬取过程中可能遇到同一个 IP 访问过于频繁的问题，此时网站就会让我们输入验证码登录或者直接封锁 IP，这样会给爬取带来极大的不便。

使用代理隐藏真实的 IP，让服务器误以为是代理服务器在请求自己。这样在爬取过程中通过不断更换代理，就不会被封锁，可以达到很好的爬取效果。

### 1.5.4 代理分类

代理分类时，既可以根据协议区分，也可以根据其匿名程度区分。

（1）根据协议区分

FTP代理服务器：主要用于访问 FTP 服务器，一般有上传、下载以及缓存功能，端口一般为 21、2121 等。

HTTP代理服务器：主要用于访问网页，一般有内容过滤和缓存功能，端口一般为 80、8080、3128 等。

SSL/TLS代理：主要用于访问加密网站，一般有SSL或TLS加密功能（最高支持128位加密强度），端口一般为443。

RTSP代理：主要用于访问Real流媒体服务器，一般有缓存功能，端口一般为554。

Telnet代理：主要用于 telnet 远程控制（黑客入侵计算机时常用于隐藏身份），端口一般为 23。

POP3/SMTP代理：主要用于 POP3/SMTP 方式收发邮件，一般有缓存功能，端口一般为 110/25。

SOCKS代理：只是单纯传递数据包，不关心具体协议和用法，所以速度快很多，一般有缓存功能，端口一般为 1080。SOCKS 代理协议又分为 SOCKS4 和 SOCKS5，前者只支持 TCP，而后者支持 TCP 和 UDP，还支持各种身份验证机制、服务器端域名解析等。简单来说，SOCK4 能做到的 SOCKS5 都可以做到，但 SOCKS5 能做到的 SOCK4 不一定能做到。

（2）根据匿名程度区分

高度匿名：会将数据包原封不动地转发，在服务端看来就好像真的是一个普通客户端在访问，而记录的 IP 是代理服务器的 IP。

普通匿名：会在数据包上做一些改动，服务端上有可能发现这是个代理服务器，也有一定几率追查到客户端的真实 IP。代理服务器通常会加入的 HTTP 头有 HTTP\_VIA 和 HTTP\_X\_FORWARDED\_FOR。

透明代理：不但改动了数据包，还会告诉服务器客户端的真实 IP。这种代理除了能用缓存技术提高浏览速度，能用内容过滤提高安全性之外，并无其他显著作用，最常见的例子是内网中的硬件防火墙。

间谍代理：指组织或个人创建的用于记录用户传输的数据，然后进行研究、监控等目的的代理服务器。

### 1.5.5 常见代理设置

（1）使用网上的免费代理

最好使用高匿代理，另外可用的代理不多，需要在使用前筛选一下可用代理，也可以进一步维护一个代理池。

（2）使用付费代理

互联网上存在许多代理商，可以付费使用，质量比免费代理好很多。

（3）ADSL拨号

拨一次号换一次 IP，稳定性高，也是一种比较有效的解决方案。

# 第2章 基本库的使用

学习爬虫，最初的操作便是**模拟浏览器向服务器发出请求**，那么我们需要从哪个地方做起呢？请求需要我们自己来构造吗？需要关心请求这个数据结构的实现吗？需要了解 HTTP、TCP、IP 层的网络传输通信吗？需要知道服务器的响应和应答原理吗？

可能你不知道无从下手，不过不用担心，Python 的强大之处就是提供了功能齐全的类库来帮助我们完成这些请求。最基础的 HTTP 库有 urllib、httplib2、requests、treq 等。

拿 urllib 这个库来说，有了它，我们只需要关心请求的链接是什么，需要传的参数是什么以及可选的请求头设置就好了，不用深入到底层去了解它到底是怎样传输和通信的。有了它，两行代码就可以完成一个请求和响应的处理过程，得到网页内容，是不是感觉方便极了？

接下来，就让我们从最基础的部分开始了解这些库的使用方法吧。

## 2.1 使用urllib

urllib是python内置的HTTP请求库。在 Python 2 中，有 urllib 和 urllib2 两个库来实现请求的发送。而在 Python 3 中，已经不存在 urllib2 这个库了，统一为 urllib，其官方文档链接为：<https://docs.python.org/3/library/urllib.html>。

首先，了解一下 urllib 库，它是 Python 内置的 HTTP 请求库，也就是说不需要额外安装即可使用。它包含如下 4 个模块。

request：它是最基本的HTTP请求模块，可以用来模拟发送请求就像在浏览器里输入网址然后回车一样，只需要给库方法传入URL及额外的参数，就可以实现这个过程。

error:异常处理模块，如果出现请求错误，我们可以捕获这些异常，然后进行重试或其他操作以保证程序不会意外终止。

parse:一个工具模块，提供了URL处理方法，比如拆分，解析，合并等。

robotparser:主要用来识别网站的robots.txt文件，然后判断哪些网站可以爬，哪些网站不可以爬，它其实用的最少。

### 2.1.1 发送请求

urllib.request模块提供了**最基本的HTTP请求的方法**，利用它可以模拟浏览器的一个请求的发起过程，同时它带有处理授权验证（authentication）、重定向（redirect）、浏览器Cookies以及其他内容。

注意：urllib.request里有哪些关键的方法？

**（1）urlpen()方法**

使用urllib的request模块，可以实现请求的发送并得到响应。

urllib.request模块里的urlopen()方法：

**from urllib.request import urlopen**

用法：利用2行代码就实现了对Python官网源代码的抓取。

**import** urllib.request

response = urllib.request.urlopen('https://www.python.org')  
print(response.read().decode('utf-8'))  
print(type(response)) # <class 'http.client.HTTPResponse'>

注意:这个reponse对象还包含了许多属性和方法。具体有哪些了？用法？

其中response对象是一个HTTPResponse类型的对象，包含read()、readinto()、getheader(name)、getheaders()、fileno()等方法以及msg、version、status、debuglever、closed等属性。

参数：

urlopen(url, data=None, timeout=socket.\_GLOBAL\_DEFAULT\_TIMEOUT,  
 \*, cafile=None, capath=None, cadefault=False, context=None)

参数说明：

data:

timeout:

其他参数：

**（2）Request类**

利用urlopen()方法可以实现最基本请求的发起，但这几个简单的参数不足以构建一个完整的请求，如果请求中需要加入Headers信息，就需要使用Requset类来解决。

urllib.request模块里的Reques类：

**from urllib.request import Request**

用法：

**import** urllib.request  
  
request = urllib.request.Request('http://python.org')  
response = urllib.request.urlopen(request)  
print(response.read().decode('utf-8'))  
print(type(request)) # <class 'urllib.request.Request'>

注意：这个request对象包含了哪些属性和方法？举例说明？

参数：

Request(url, data=None, headers={},origin\_req\_host=None, unverifiable=False,method=None)

参数说明：

url:

data:

headers:

origin\_req\_host:

unverifiable:

method:

**（3）BaseHandler类**

在上面的2个过程中，我们虽然可以构造请求，但是对于一些高级的操作（如Cookie的处理，代理设置等）就麻烦了。

这里提到了强大的Handler工具。可以被理解成各种处理器，有专门处理验证的，有处理Cookies的，有处理代理设置的。利用它们可以做到HTTP请求中的所有事情？

urllib.request模块里的BaseHandler类，它是所有其他Handler类的父类，它提供了最基本的方法，如default\_open()、protocol\_request()等。有如下Handler:

HTTPDefaultErrorHandler:用于处理HTTP响应错误，错误都会抛出HTTPError类型的异常。

HTTPRedirectHandler:用于处理重定向。

HTTPCookieProcessor:用于处理Cookie。

ProxyHandler:用于处理代理，默认代理为空。

HTTPPasswordMgr:用于管理密码，它维护了用户名和密码的表。

HTTPBasicAuthHandler:用于管理验证，如果一个链接打开时需要认证，那么可以用它来解决认证问题。

另一个比较重要的类是OpenerDirector，我们可以称为Opener。我们之前用的urlopen()这个方法，实际上就是urllib为我们提供的Opener。

那么为什么引入Opener呢？因为需要实现更高级的功能，之前使用的Request和urlopen()相当于类库为你封装好了极其常用的请求方法，利用它们可以实现基本的请求，但是现在不一样了，我们需要实现更高级的功能，所以需要深入一层进行配置，使用更底层的实例来完成操作，所以就用到了Opener。

Opener可以使用open()方法，返回的类型和urlopen()如出一辙。利用Handler来构建Opener。

1) 验证

有些网站在打开时会弹出提示框，提示输入用户名和密码，验证成功后才能查看页面。

**from** urllib.request **import** HTTPPasswordMgrWithDefaultRealm, HTTPBasicAuthHandler, build\_opener  
**from** urllib.error **import** URLError  
  
username = 'jack'  
password = '123456'  
url = 'http://localhost:5000/'  
  
p = HTTPPasswordMgrWithDefaultRealm()  
p.add\_password(**None**, url, username, password)  
auth\_handler = HTTPBasicAuthHandler(p)  
opener = build\_opener(auth\_handler)  
  
**try**:  
 result = opener.open(url)  
 html = result.read().decode('utf-8')  
 print(html)  
**except** URLError **as** e:  
 print(e.reason)

这里首先实例化HTTPBasicAuthHandler对象，其参数是HTTPPasswordMgrWithDefaultRealm对象，它利用add\_password()添加进去用户名和密码，这样就构建了一个处理验证的Handler。

接下来，利用这个Handler并使用build\_opener()方法构建了一个Opener，这个Opener在发送请求的时候就相当于已经验证成功了。

接下来，利用Opener的open()方法打开链接，就可以完成验证了。这里获取的结果便是验证后的源码内容。

2) 代理

在做爬虫时，免不了使用代理。如果要添加代理，可以这样做：

**from** urllib.error **import** URLError  
**from** urllib.request **import** ProxyHandler, build\_opener  
  
proxy\_handler = ProxyHandler({  
 'http': 'http://127.0.0.1:9743',  
 'https': 'https://127.0.0.1:9743'  
})  
  
opener = build\_opener(proxy\_handler)  
  
**try**:  
 response = opener.open('https://www.baidu.com')  
 print(response.read().decode('utf-8'))  
**except** URLError **as** e:  
 print(e.reason)

这里我们在本地搭建了一个代理，它运行在9743端口上。

这里使用了ProxyHandler，其参数是一个字典，键名是协议类型（比如HTTP或者HTTPS等），键值时代理链接，可以添加多个代理。

然后利用这个Handler及build\_opener()方法构造一个Opener()，之后发送请求即可。

3) Cookies

Cookies的相关处理。

**import** http.cookiejar  
**import** urllib.request  
  
cookie = http.cookiejar.CookieJar()  
handler = urllib.request.HTTPCookieProcessor(cookie)  
opener = urllib.request.build\_opener(handler)  
response = opener.open('http://www.baidu.com')  
  
**for** item **in** cookie:  
 print(item.name + '=' + item.value)

首先我们必须声明一个CookieJar对象。接下来利用HTTPCookieProcessor来构建一个Handler，最后利用build\_openr()方法构建出Opener，执行open()函数即可。

输出成文件格式：

**import** http.cookiejar  
**import** urllib.request  
  
filename = 'cookies.txt'  
# cookie = http.cookiejar.MozillaCookieJar(filename) # 保存格式1  
cookie = http.cookiejar.LWPCookieJar(filename) # 保存格式2  
handler = urllib.request.HTTPCookieProcessor(cookie)  
opener = urllib.request.build\_opener(handler)  
response = opener.open('http://www.baidu.com')  
  
cookie.save(ignore\_discard=**True**, ignore\_expires=**True**)

这时候将CookieJar对象换成MozillaCookieJar对象，它在生成文件时会用到，是CookieJar的子类，可以用来处理Cookies和文件相关的事件，比如读取和保存Cookies，可以将Cookies保存成Mozilla型浏览器的Cookies格式。运行之后会生成一个txt格式的文件。

另外，LWPCookieJar对象也可以读取和保存Cookies，但是保存的格式和MozillaCookieJar不一样，它会保存程libwww-per(LWP)格式的Cookies文件。

读取并利用：

**import** http.cookiejar  
**import** urllib.request  
  
cookie = http.cookiejar.LWPCookieJar()  
cookie.load('cookies.txt', ignore\_discard=**True**, ignore\_expires=**True**)  
handler = urllib.request.HTTPCookieProcessor(cookie)  
opener = urllib.request.build\_opener(handler)  
response = opener.open('https://www.baidu.com')  
  
print(response.read().decode('utf8'))

可以看到，这里调用load()方法来读取本地的Cookies文件，获取到了Cookies的内容。不过前提是我们生成了LWPCookieJar格式的Cookies，并保存成文件，然后读取Cookies之后使用同样的方法构建Handler和Opener即可完成操作。

运行结果正常的话，会生成百度网页的源代码。

### 2.1.2 处理异常

前面我们了解了请求的发送过程，但是在网络不好的情况下，如果出现了异常，该怎么办？这时候如果不处理这些异常，程序很可能因报错而终止运行，所以异常处理还是非常必要的。

urllib的error模块定义了由request模块产生的异常，如果出现了异常，request模块便会抛出error模块中定义的异常。

(1) URLError

URLError类来自于urllib库的error模块，它继承自OSError类，是error异常类模块的基类，由request模块产生的异常都可以通过捕获这个类来处理。

(2) HTTPError

HTTPError是URLError的子类，专门用来处理HTTP请求的错误，比如认证失败等。它有如下3个属性：

code:返回HTTP状态码，比如404表示网页不存在，500表示服务器内部错误等。

reason:同父类一样，用于返回错误的原因。

headers:返回请求头。

### 2.2.3 解析链接

urllib模块里面提供的parse模块，它定义了处理URL的标准接口，例如实现URL各部分的抽取、合并以及链接转换。它支持如下协议的URL处理：file、ftp、gopher、hdl、http、https imap、mailto、mms、news、nntp、prospero、rsync、rtsp、stspu、sftp、sip、sips、snews、svn、svn+ssh、telnet、wais等。

\_\_all\_\_ = ["urlparse", "urlunparse", "urljoin", "urldefrag",  
 "urlsplit", "urlunsplit", "urlencode", "parse\_qs",  
 "parse\_qsl", "quote", "quote\_plus", "quote\_from\_bytes",  
 "unquote", "unquote\_plus", "unquote\_to\_bytes",  
 "DefragResult", "ParseResult", "SplitResult",  
 "DefragResultBytes", "ParseResultBytes", "SplitResultBytes"]

(1) urlparse()

(2) urlunparse()

(3) urlsplit()

(4) urlunsplit()

(5) urljoin()

(6) urlencode()

(7) parse\_qs()

(8) parse\_qsl()

(9) quote()

(10) unquote()

### 2.2.4 分析Robots协议

利用urllib的robotparser模块，我们可以实现网站Robots协议的分析。

(1) Robots协议

Robots 协议也称作爬虫协议、机器人协议，它的全名叫作网络爬虫排除标准（Robots Exclusion Protocol），用来告诉爬虫和搜索引擎哪些页面可以抓取，哪些不可以抓取。它通常是一个叫作 robots.txt 的文本文件，一般放在网站的根目录下。

当搜索爬虫访问一个站点时，它首先会检查这个站点根目录下是否存在 robots.txt 文件，如果存在，搜索爬虫会根据其中定义的爬取范围来爬取。如果没有找到这个文件，搜索爬虫便会访问所有可直接访问的页面。

(2) 爬虫名称

一些常见的搜索爬虫的名称极其对应的网站

|  |  |  |
| --- | --- | --- |
| 爬虫名称 | 名称 | 网站 |
| BaiduSpider | 百度 | www.baidu.com |
| Googlebot | 谷歌 | www.google.com |
| 360Spider | 360 搜索 | www.so.com |
| YodaoBot | 有道 | www.youdao.com |
| ia\_archiver | Alexa | www.alexa.cn |
| Scooter | altavista | www.altavista.com |

(3) robotparser

了解 Robots 协议之后，我们就可以使用 robotparser 模块来解析 robots.txt 了。该模块提供了一个类 RobotFileParser，它可以根据某网站的 robots.txt 文件来判断一个爬虫是否有权限来爬取这个网页。

## 2.2 使用requests

urllib模块有许多不便的地方，比如处理网页验证的Cookies，需要写Opener和Handler来处理。而request模块具有更强大和更方便的功能。

### 2.2.1 基本用法

urllib库中的urlopen()方法实际上是以GET方式请求网页，而requests中相应的方法就是get()方法。

**import** requests  
  
r = requests.get('http://www.baidu.com')  
  
print(type(r)) # <class 'requests.models.Response'>  
print(r.status\_code) # 200  
print(type(r.text)) # <class 'str'>  
print(r.text)  
print(r.cookies)

这里我们调用了get()方法实现与urlopen()相同的操作，得到了一个Response对象，然后分别输出了Response的类型，状态码，响应体的类型，内容以及Cookies。

其他的请求方法，依然可以通过一句话来解决：

r = requests.post('http://www.baidu.com')  
r = requests.put('http://www.baidu.com')  
r = requests.delete('http://www.baidu.com')  
r = requests.head('http://www.baidu.com')  
r = requests.options('http://www.baidu.com')  
r = requests.patch('http://www.baidu.com')

requests.models.Response对象的一些属性和方法：

['apparent\_encoding', 'close', 'connection', 'content', 'cookies', 'elapsed', 'encoding', 'headers', 'history', 'is\_permanent\_redirect', 'is\_redirect', 'iter\_content', 'iter\_lines', 'json', 'links', 'next', 'ok', 'raise\_for\_status', 'raw', 'reason', 'request', 'status\_code', 'text', 'url']

（1）GET请求

首先，构建一个最简单的GET请求，请求的链接为<http://httpbin.org/get>，该网站会判断如果客户端发起的是GET请求，就会返回相应的请求信息。

GET请求的返回结果中包含请求头，URL，IP等信息。

那么对于GET请求如何附加额外的信息呢？可以增加一个参数params:

**import** requests  
  
data = {  
 'name': 'jack',  
 'age': 18  
}  
r = requests.get('http://httpbin.org/get', params=data) # 附加参数  
print(r.text)  
print(type(r.text)) # <class 'str'>  
  
print(r.json())  
print(type(r.json())) # <class 'dict'>

通过运行结果判断可知，请求的链接自动变成了<http://httpbin.org/get?name=jack&age=18>

另外，网页的返回类型实际上是str类型，但是它特殊，是一个JSON格式的。所以，如果想直接解析返回结果，得到一个字典格式的话，可以调用json方法。

运行结果发现，调用json()方法后，就可以将返回结果是JSON格式的字符串转换为字典。

但需要注意的是，如果返回结果不是JSON格式的，就会出现解析错误，抛出json.decoder.JSONDecoderError。

抓取网页：

**import** requests  
**import** re  
  
headers = {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.77 Safari/537.36'  
}  
  
r = requests.get('https://www.zhihu.com/explore', headers=headers)  
pattern = re.compile('explore-feed.\*?question\_link.\*?>(.\*?)</a>', re.S)  
titles = re.findall(pattern, r.text)  
print(titles)

这里加入了headers信息，其中包含了User-Agent字段信息，也就是浏览器标识信息。如果不加这个，知乎会禁止抓取，可以试试在headers头中加不加User-Agent字段的影响。接下来，使用了最基础的正则表达式来匹配所有问题内容。

抓取二进制数据：

上面抓取的实际是一个HTML文档，如果想要抓取图片，音频，视频等文件，该怎么办？

图片，音频，视频这些文件本质上都是由二进制码组成的，由于有特性的保存格式和对应的解析方式，我们才可以看到这些形形色色的多媒体。

**import** requests  
  
r = requests.get('https://github.com/favicon.ico')  
# print(r.text)  
# print(r.content)  
  
**with** open('favicon.ico', 'wb') **as** f:  
 f.write(r.content)

这里抓取的是GitHub网站的站点图标，打印了Response对象的两个属性，一个是text，另一个是content。接下来，我们将刚刚提取的图片保存下来，二进制数据使用wb标识符。

（2）POST请求

首先，构建一个最简单的POST请求，请求的链接为<http://httpbin.org/post>，该网站会判断如果客户端发起的是POST请求，就会返回相应的请求信息。

**import** requests  
  
data = {  
 'name': 'jack',  
 'age': 18  
}  
  
r = requests.post('http://httpbin.org/post', data=data)  
print(r.text)

（3）响应

发送请求之后，得到的自然就是响应。在上面的例子中，我们使用了text和content获取了响应的内容。此外，还有其他很多属性和方法获取其他信息，比如状态码，响应头，Cookies等。示例如下：

**import** requests  
  
r = requests.get('http://www.janshu.com')  
print(type(r.status\_code), r.status\_code)  
print(type(r.headers), r.headers)  
print(type(r.cookies), r.cookies)

### 2.2.2 高级用法

（1）文件上传

我们知道requests可以提交一些数据。假如有些网站需要上传文件，可以用它来实现。

**import** requests  
  
files = {'file': open('favicon.ico', 'rb')}  
r = requests.post('http://httpbin.org/post', files=files)  
  
print(r.text)

（2）Cookies

获取Cookies

**import** requests  
  
r = requests.get('https://www.baidu.com')  
print(r.cookies)  
  
**for** key, value **in** r.cookies.items():  
 print(key + '=' + value)

维持登陆状态

（3）会话维持

（4）SSL证书验证

（5）代理设置

（6）超时设置

（7）身份验证

（8）Prepared Request

## 2.3 正则表达式

## 2.4 爬取猫眼电影排行

# 第3章 解析库的使用

## 3.1 使用XPath

## 3.2 使用Beautiful Soup

## 3.3 使用pyquery

# 第4章 数据存储

## 4.1 文件存储

### 4.1.1 TXT文本存储

### 4.1.2 JSON文件存储

### 4.1.3 CSV文件存储

## 4.2 关系型数据库存储

### 4.2.1 MySQL的存储

## 4.3 非关系型数据库的存储

### 4.3.1 MongoDB存储

### 4.3.2 Redis存储

# 第5章Ajax数据爬取

## 5.1 什么是Ajax

## 5.2 Ajax分析方法

## 5.3 Ajax结果提取

## 5.4 分析Ajax爬取今日头条街拍美图

# 第6章 动态渲染页面爬取

## 6.1 Selenium的使用

## 6.2 Splash的使用

## 6.3 Splash负载均衡配置

## 6.4 使用Selenium爬取淘宝商品

# 第7章 验证码的识别

## 7.1 图形验证码的识别

## 7.2 极验滑动验证码的识别

## 7.3 点触验证码的识别

## 7.4 微博宫格验证码的识别

# 第8章 代理的使用

## 8.1 代理的设置

## 8.2 代理池的维护

## 8.3 付费代理的使用

## 8.4 ADSL拨号代理

## 8.5 使用代理爬取微信公众号文章

# 第9章 模拟登陆

## 9.1 模拟登陆并爬取GitHub

## 9.2 Cookies池的搭建

# 第10章 App的爬取

## 10.1 Charles的使用

## 10.2 mitmproxy的使用

## 10.3 mitmproxy爬取“得到”App电子书信息

## 10.4 Appium的基本使用

## 10.5 Appium爬取微信朋友圈

## 10.6 Appium+mitmdump爬取京东商品

# 第11章 pyspider的使用

## 11.1 pyspider框架介绍

## 11.2 pyspider的基本使用

## 11.3 pyspider用法详解

# 第12章 Scrapy框架的使用

## 12.1 Scrapy框架介绍

## 12.2 Scrapy入门

## 12.3 Selector的用法

## 12.4 Spider的用法

## 12.5 Downloader Middleware的用法

## 12.6 Spider Middleware的用法

## 12.7 Item Pipeline的用法

## 12.8 Scrapy对接Selenium

## 12.9 Scrapy对接Splash

## 12.10 Scrapy通用爬虫

## 12.11 Scrapyrt的使用

## 12.12 Scrapy对接Docker

## 12.13 Scrapy爬取新浪微博

# 第13章 分布式爬虫

## 13.1 分布式爬虫原理

## 13.2 Scrapy-Redis源码解析

## 13.3 Scrapy分布式实现

## 13.4 Bloom Filter的对接

# 第14章 分布式爬虫的部署

## 14.1 Scrapyd分布式部署

## 14.2 Scrapyd-Client的使用

## 14.3 Scrapyd对接Docker

## 14.4 Scrapyd批量部署

## 14.5 Cerapy分布式管理