第3章

Web应用架构与协议

Web服务器存储了Web页面，按照一定的方式响应客户端的请求。理解Web服务器的工作原理，对于掌握和设计网络爬虫具有重要意义。本章简述Web服务器的应用架构，重点对网站页面文件的组织与内容生成、HTTP、状态保持技术以及Robots协议进行详细介绍。

3.1 常用的Web服务器软件

3.1.1 流行的Web服务器软件

常见的Web服务器软件有Apache、IIS(Internet Information Server)、Nginx、Lighttpd、Zeus、Resin、Tomcat等。

Apache是使用范围很广的Web服务器软件，是Apache软件基金会的一个开放源代码的跨平台网页服务器。它可以运行在几乎所有主流的计算机平台上。Apache的特点是简单、速度快、性能稳定。它支持基于IP或域名的虚拟主机，支持代理服务器，支持安全套接字层(Secure Socket Layer，SSL)等。目前，互联网网站主要使用它作为静态资源服务器，也可以作为代理服务器转发请求，同时结合Tomcat等Servlet容器处理JSP等动态网页。Apache Web服务器软件网站页面如图3-1所示。

Nginx是一个高性能的HTTP和反向代理服务器，国内使用Nginx作为Web服务器的网站也越来越多，其中包括新浪博客、网易新闻、搜狐博客等门户网站频道，在3万以上访问次数的高并发环境下，Nginx的处理能力相当于Apache的10倍。Nginx Web服务器软件网站页面如图3-2所示。

图 3-1　Apache Web 服务器软件网站页面

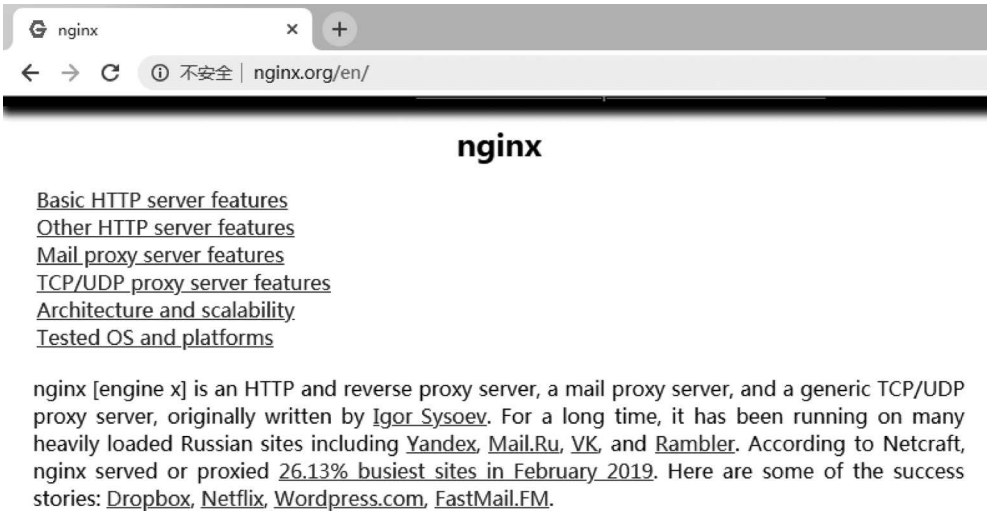

图 3-2　Nginx Web 服务器软件网站页面

IIS(Internet 信息服务)是微软公司主推的 Web 服务器,IIS 与 Windows Server 完全集成在一起,因此用户能够利用 Windows Server 和新技术文件系统(New Technology File System,NTFS)内置的安全特性建立强大、灵活且安全的 Internet 和 Intranet 站点。IIS Web 服务器软件网站页面如图 3-3 所示。

Tomcat 是 Apache 软件基金会的 Jakarta 项目中的一个核心项目,由 Apache、Sun 和其他一些公司及个人共同开发。因为 Tomcat 技术先进、性能稳定,而且免费,所以深受 Java 爱好者的喜爱,并得到了部分软件开发商的认可,是目前比较流行的 Web 应用服务器。Tomcat Web 服务器软件网站页面如图 3-4 所示。

3.1.2　在 Python 中配置 Web 服务器

Python 3 提供了小型 Web 服务器软件的功能,可以很方便地进行 Web 页面的开发和测试。以 Windows 操作系统为例,具体过程如下。

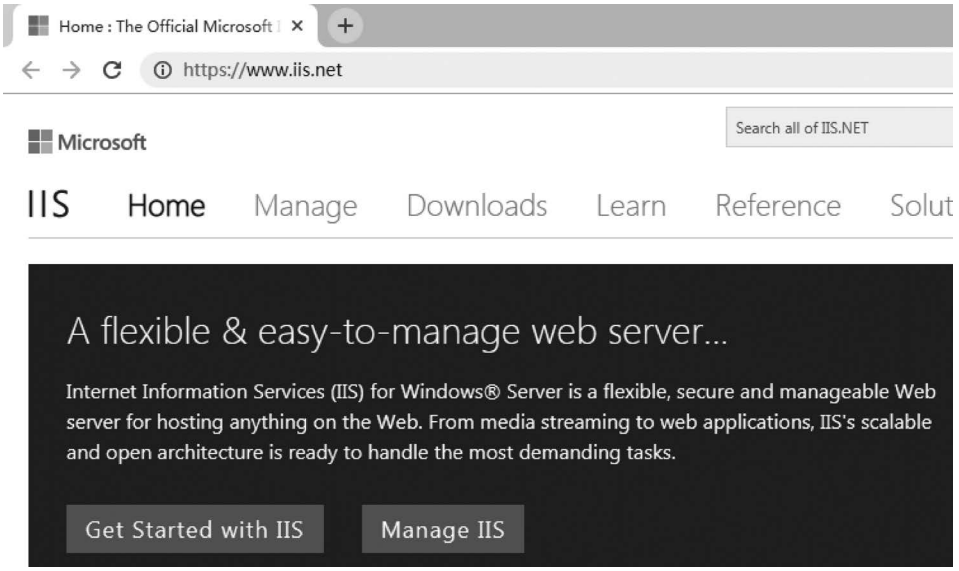

图 3-3 IIS Web 服务器软件网站页面

图 3-4 Tomcat Web 服务器软件网站页面

(1) 通过执行 cmd 命令进入 Windows 控制台。

(2) 切换到 Web 服务器的虚拟根目录,也就是存放网页的根目录位置。

(3) 执行如下命令。

```
python -m http.server 端口号
```

以下是一个例子,假设虚拟根目录是 E:\xxx,则进入控制台后执行以下命令。

```
C:\> e:
E:\> cd xxx
E:\xxx > python -m http.server 8080
```

那么,启动成功后控制台显示如下。

```
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

之后，可以在浏览器中输入 http://localhost:8080/浏览 E:\xxx 中的页面和文件，在浏览的过程中控制台默认输出每个点击访问记录，样例如下。

```
127.0.0.1 - - [05/Nov/2019 15:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2019 15:43:36] "GET /1.html HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2019 15:43:46] "GET /20.html HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2019 15:44:03] "GET /4.html HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2019 15:44:12] "GET /ch.txt HTTP/1.1" 200 -
```

3.2　Web 服务器的应用架构

基于 Web 的互联网应用都离不开 Web 服务器，在门户网站、网络论坛、电子商务网站等典型应用中，核心部件都是 Web 服务器。Web 服务器为互联网用户提供页面信息访问服务，基本的访问方式包括浏览信息、发布信息等形式。

3.2.1　典型应用架构

从应用架构的角度看，Web 服务器与客户端构成一种 Client/Server 的技术体系。在实际应用中需要考虑网站内容的可管理性、网站的并发能力、容错能力以及网站的可维护性等许多问题。针对这些问题有不同的解决方案，由此产生不同的应用架构。4 种典型的应用架构描述如下。

1. Client/Server

最简单的 Web 应用架构是单纯的 Client/Server 架构，它也是其他架构的基础，如图 3-5 所示。其中，客户端可以是各种浏览器，也可以是爬虫程序。

在这种架构中，一方面，Web 服务器作为存储器，各种 HTML 文件可以直接存储在 Web 服务器的硬盘上，根据用户的请求情况再访问这些文件；另一方面，Web 服务器也是一个执行机构，能够处理用户的请求。在网络爬虫技术中，这种应用架构适用于静态网页的处理，每个静态网页对应硬盘上的一个文件。

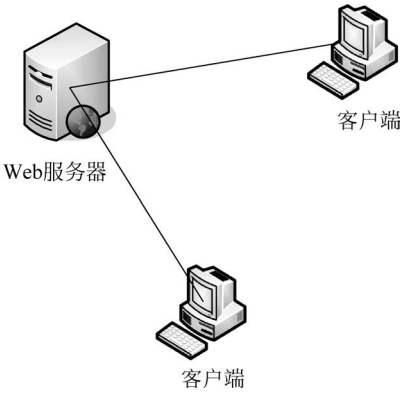

图 3-5　Client/Server 架构

2. Client/Server/Database

在很多 Web 应用中，并不能简单地从 Web 服务器的硬盘上读取 HTML 文件内容推送给用户，而是需要读取后台数据库或访问其他服务器。相应地，在 Web 应用架构中就必须增加数据库服务

器,如图 3-6 所示。

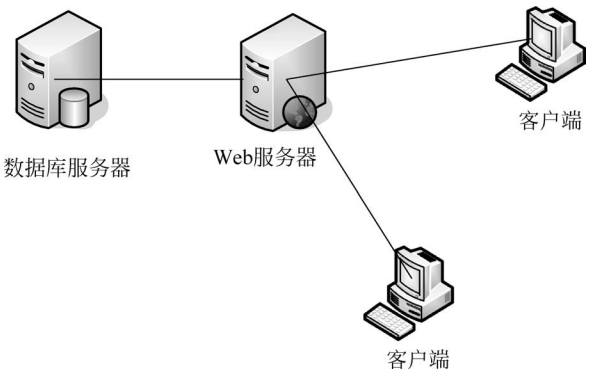

图 3-6 Client/Server/Database 架构

在这种架构中,Web 服务器上的 HTML 文件中通常存在一些动态脚本,这些脚本在用户请求时由 Web 服务器执行。在执行过程中访问数据库,获取数据访问结果,Web 服务器再将执行结果编码成 HTML,推送给客户端。在网络爬虫技术中,这种架构支持了动态网页的实现。也就是说,网页中的主体内容是来自数据库或其他服务器,而不是直接存在 Web 服务器的 HTML 文件中。因此,动态网页的采集会消耗更多的服务器资源,包括 Web 服务器内存、网络带宽以及数据库服务器的连接,爬虫采集策略优化时要考虑这些因素。

3. Web 服务器集群

互联网 Web 服务器提供了开放式服务,可能会有大量用户并发访问的情况出现。在这种情况下,为了保证用户访问的体验度和容错性,Web 服务器通常需要进行高可用和负载均衡设计。

负载均衡就是根据某种任务均衡策略把客户端请求分发到集群中的每台服务器上,让整个服务器群均衡地处理网站请求。因此,集群是这种应用架构的典型特征和核心之一,通常用多台 Web 服务器构成一个松耦合系统,这些服务器之间通过网络通信实现相互协作,共同承载所有客户端的请求压力。如图 3-7 所示的应用架构就是这种集群架构,整个集群对外来看是一台 Web 服务器。

在这种架构中,网络爬虫每次连接到网站的 IP 地址可能并不是固定的,因此如果网络爬虫进行了 DNS 缓存优化设计,就应当考虑到 Web 服务器集群的这种具体情况。

4. 虚拟主机

虚拟主机是另一种常见的 Web 应用架构,它是指在一台服务器上配置多个网站,使得每个网站看起来具有独立的物理计算机。虚拟主机的实现方法有以下 3 种。

(1) 基于 IP 地址的方法。在服务器上绑定多个 IP,配置 Web 服务器,把网站绑定在不同的 IP 上。当客户端或爬虫访问不同的 IP 地址时就得到不同网站的响应。

(2) 基于端口的方法。不同网站共享一个 IP 地址,但是通过不同的端口实现对不同

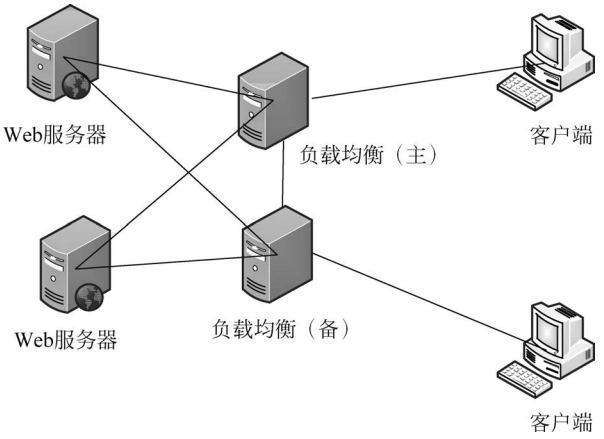

图 3-7　Web 服务器集群

网站的访问。这时客户端访问的 URL 的形式为 http://hostname:port/，需要指定端口。

（3）基于主机名的方法。设置 DNS 将多个域名解析到同一个 IP 地址上，IP 地址对应的服务器上配置 Web 服务端，添加多个网站，为每个网站设定一个主机名。因为 HTTP 访问请求头里包含有主机名（即 host 属性）信息，所以当 Web 服务器收到访问请求时就可以根据不同的主机名访问不同的网站。

3.2.2　Web 页面的类型

从 Web 网页的组成结构来看，一个标准的网页一般包含 4 部分，即内容、结构、表现效果和行为。内容是网页中直接传达给阅读者的信息，包括文本、数据、图片、音/视频等；结构是指 Web 页面的布局，对内容进行分类使之更具有逻辑性，符合用户的浏览习惯；表现效果是指对已经结构化的内容进行视觉感官上的渲染，如字体、颜色等；行为是指网页内容的生成方式。

根据 Web 页面组成结构中的信息内容的生成方式，可以将 Web 页面分为静态页面、动态页面、伪静态页面三大类。静态页面随着互联网的出现而出现，虽然目前有大量的交互式动态页面能很好地提升用户体验，但是由于静态页面除了 Web 服务外，不需要其他服务的支持，对服务器的资源消耗少，所以这种类型的页面在现阶段仍然被广泛使用。动态页面一般需要数据库等其他计算、存储服务的支持，因此需要消耗更多的服务端资源。Web 服务器在响应爬虫命令请求时，不管是静态页面还是动态页面，服务器返回给爬虫的信息一般都是封装成为 HTML 格式。因此，对于页面解析器，在处理 Web 服务器响应信息时，只要处理 HTML 格式的内容即可。

静态页面以 HTML 文件的形式存在于 Web 服务器的硬盘上，其内容和最终显示效果是事先设计好的，在环境配置相同的终端上的显示效果是一致的，并且这些内容、表现效果等由且仅由 HTML、JavaScript、CSS 等语言控制。因此，静态页面的内容、结构和表现效果是固定的，其行为也比较简单，即在制作网页时直接保存到硬盘文件。

从实际应用的角度来看,目前互联网上大部分都是动态页面。动态页面是相对静态页面而言的,具有明显的交互性,能根据用户的要求和选择动态改变。动态页面的主要特征如下。

(1) 页面内容是可变的,不同人、不同时间访问页面时,显示的内容可能不同。

(2) 页面结构也是允许变化的,但为了提升用户浏览的体验,结构一般不会频繁改变。

(3) 在表现效果上,页面中不同部分的效果会随着内容的变化而变化。

(4) 页面行为是区别于静态页面最主要的特征,动态页面并不是直接把内容存储到文件中,而是要进一步执行内容生成步骤,通常的方式有访问数据库等。

除了静态页面和动态页面外,还有一类页面称为伪静态页面,这种页面的出现是为了增强搜索引擎爬虫的友好界面。伪静态页面是以静态页面展现出来,但实际上是用动态脚本来处理的。为了达到这个目的,伪静态页面技术通常采用 404 错误处理机制或 rewrite 技术实现将静态的 URL 页面请求转换成内部的动态访问,再将结果以同样的 URL 返回给用户。

伪静态页面的 URL 本质上可以理解成不带“=”的参数,是将参数转换为 URL 的子目录或文件名的一部分。例如,某论坛中同一个帖子的伪静态页面地址及其对应的动态页面地址分别如下。

```
http://****.com/htm_data/7/1985527.html
http://****.com/read.php?fid=7&tid=1985527
```

其中,伪静态页面地址的 htm_data 后面的 7 以及 1985527 分别与动态页面地址的 fid=7 和tid=1985527 对应。又如,股吧论坛一个帖子对应的 URL 是 https://guba.eastmoney.com/news/300059/948261749.html,其后缀名为 html,该 URL 实际上是一个伪静态页面,因为页面中的帖子及其对应的回帖内容必定是动态生成的。当服务器收到伪静态页面的 URL 地址请求后提取出 URL 中相应的部分,如本例中的 300059 和 948261749,并作为动态参数构造动态请求,在执行完成后将结果返回给客户端。

3.2.3 页面文件的组织方式

大量的 Web 页面文件在 Web 服务器中的组织管理方式对于提升页面的可维护性是非常重要的。下面以 Tomcat 为例说明服务器的页面文件管理方式。

如图 3-8 所示,在 D 盘的 Tomcat 安装目录下有子目录 webapps,该子目录是 Tomcat 规定的虚拟根目录。在该子目录下有 aaa 和 bbb 两个子目录,而 aaa 下还有一个子目录 images。其中,aaa 下有两个文件 a1.html 和 a2.html; bbb 下有一个文件 b1.html; images 子目录下有 3 个文件,分别为 p1.gif、p2.gif、p3.gif。

假设按照 Tomcat 的默认端口 8080,那么在浏览器中访问 a1.html 的 URL 为

```
http://127.0.0.1:8080/aaa/a1.html
```

其相应的结果如图 3-9 所示，页面中的图片在 a1.html 中是通过< img src="images/p1.gif">来获得。这里的 images/p1.gif 就表示在当前目录中访问 images 子目录下的 p1.gif，而由于访问的是 a1.html 文件，当前目录就是 aaa。

图 3-8　Tomcat 的目录结构

图 3-9　a1.html 的访问方式及效果

下面的 HTML 代码则展示了 a1.html 页面中超链接的 3 种写法。

（1）采用相对链接，访问 a2.html。

```
<p><img SRC="images/p1.gif"><a href="a2.html">a2</a></p>
```

（2）采用相对链接，访问 b1.html，..表示上级目录，此处即为虚拟根目录。

```
<p><img SRC="images/p2.gif"><a href="..\bbb\b1.html">b1</a></p>
```

（3）采用 http 开始的完整 URL 绝对链接，访问 b1.html。

```
<p><img SRC="images/p3.gif"><a href="http://127.0.0.1:8080\bbb\b1.html">b1</a></p>
```

从这个例子可以看出，对于爬虫，在获取 a1.html 页面之后要寻找其中的 href 超链接。对于绝对链接，只需要把 href=后面的字符串提取出来即可；而对于相对链接，没有完整的 http，单纯从这个 href 所指定的链接无法知道其真正的结果，需要进行超链接的转换。

为了简化相对路径的转换，在 HTML 中提供了< base >标签，用来指定页面中所有超链接的基准路径。例如，如果在 a1.html 中增加如下< base >标签：

```
<base href="http://127.0.0.1:8080/aaa/" />
```

那么就意味着该文件中的所有超链接都以根目录下的 aaa 子目录为当前目录。因此，作为爬虫程序，也应当检查 HTML 文档中是否存在< base >标签。如果不存在，则需要自行解析 href 的指向，并转换为绝对链接。

扫一扫
视频讲解

3.3 Robots 协议

3.3.1 Robots 协议的来历

普通的 Web 网站提供了开放式的空间,用户只要通过浏览器即可直接访问,大部分新闻网站、机构主页都属于这种形式。另一种 Web 网站页面则提供有限的开放式空间,需要经过身份识别后进一步访问更多页面信息,最常用的方法是要求用户输入用户名和口令信息,验证通过后访问 Web 页面。典型的这种 Web 网站页面是在线购物、社交媒体等网站中的个人相关信息页面。

对于搜索引擎爬虫,可以畅通无阻地采集第一种 Web 页面,也可以在适当配置后以动态页面的方式访问第二种 Web 页面。因此,不管哪种类型的 Web 页面,最终都可以被搜索引擎收录而永久存储,并提供给用户任意搜索。这个结果在一定程度上增加了网站被其他人了解的可能性,为网站带来一定的流量增加。但是,某些页面可能是临时性的,或者不希望永久地存储于搜索引擎中。

为了给 Web 网站提供灵活的控制方式决定页面是否能够被爬虫采集,1994 年搜索行业正式发布了一份行业规范,即 Robots 协议。Robots 协议又称为爬虫协议、机器人协议等,其全称是 Robots Exclusion Protocol,即“网络爬虫排除协议”。这一协议几乎被所有搜索引擎采纳,包括 Google、Bing、百度、搜狗等。随着互联网大数据时代的到来,目前互联网上除了搜索引擎的爬虫外还存在大量的非搜索引擎爬虫,该协议也就成为各类爬虫行为的行业规范。

3.3.2 Robots 协议的规范与实现

网站通过 Robots 协议告诉爬虫哪些页面可以抓取,哪些页面不能抓取。Robots 协议指定了某种标识的爬虫能够抓取的目录或不能抓取的目录,也就是访问许可策略。这些访问许可的定义写在一个名为 robots.txt 的文件中,该文件需要放在网站的虚拟根目录中。它可以公开访问,即在浏览器中打开网站后,在网站首页的地址后面添加/robots.txt,如果网站设置了访问许可,按 Enter 键就可以看到网站的 Robots 协议,即 robots.txt 文件的内容。

robots.txt 文件的具体约定如下。

(1) 文件中包含一个或多个记录,每个记录由一个或多个空白行隔开。每个记录由多行组成,每行的形式为

```
<field>:<optionalspace><value><optionalspace>
```

记录指出了每个字段及其对应的值。字段名有 User-Agent、Disallow、Allow,每个记录就是由这些关键词来规定爬虫的访问许可。

在一个记录中可以有多个 User-Agent、多个 Disallow 或 Allow。

（2）User-Agent 的使用方式是 User-Agent [agent_name]，其中 agent_name 有两种典型形式，即 * 和具体的爬虫标识。例如：

```
User-Agent: *                表示所定义的访问许可适用于所有爬虫
User-Agent: Baiduspider      表示所定义的访问许可适用于标识为 Baiduspider 的爬虫
```

一些常见的爬虫标识有 Baiduspider、Baiduspider-image、Baiduspider-news、Googlebot、YoudaoBot、Sogou web spider、Sosospider、EasouSpider 等，从名字本身可以看出爬虫是属于哪个公司的。这些标识是爬虫自己确定的，并对外公开，如百度的爬虫可以在 http://baidu.com/search/spider.htm 查到。

（3）Disallow 和 Allow 的使用方法相同，只是决定了不同的访问许可。这种访问许可是针对目录、文件或页面而言的，即允许或不允许访问。Robots 协议的默认规则是一个目录如果没有显式声明为 Disallow，它是允许访问的。

下面以 Disallow 为例进行说明。

Disallow 的典型写法如下。

```
Disallow:/
Disallow:/homepage/
Disallow:/login.php
```

Disallow 指定的字段值可以是一个全路径，也可以是部分路径，任何以该字段值开始的 URL 都会被认为是不允许访问的，如 Disallow:/help 就蕴含着不允许访问/help.html 和/help/index.html 等，而 Disallow:/help/不会限制对/help.html 的访问，但是对/help/index.html 限制访问。

接下来看一些典型的例子。

1）https://www.taobao.com/robots.txt

摘录两条记录，分别表示该网站对 Bingbot 和 360Spider 的访问许可。可以看出，对于前者，除了 8 条许可访问外，其他都是不允许抓取的；而对于 360Spider，只有 3 条许可访问。

```
User-Agent: Bingbot
Allow:/article
Allow:/oshtml
Allow:/product
Allow:/spu
Allow:/dianpu
Allow:/oversea
Allow:/list
Allow:/ershou
Disallow:/

User-Agent: 360Spider
```

```
Allow:/article
Allow:/oshtml
Allow:/ershou
Disallow:/
```

2) https://www.baidu.com/robots.txt

摘录最后3条记录,前两条分别表示该网站对yisouspider和EasouSpider的访问许可。可以看出,除了8条不允许访问外,其他没有写的目录都是允许抓取的。最后一条记录表示除了前面定义的爬虫外,其他爬虫不允许访问整个网站(Disallow:/)。

```
User-Agent: yisouspider
Disallow:/baidu
Disallow:/s?
Disallow:/shifen/
Disallow:/homepage/
Disallow:/cpro
Disallow:/ulink?
Disallow:/link?
Disallow:/home/news/data/

User-Agent: EasouSpider
Disallow:/baidu
Disallow:/s?
Disallow:/shifen/
Disallow:/homepage/
Disallow:/cpro
Disallow:/ulink?
Disallow:/link?
Disallow:/home/news/data/

User-Agent: *
Disallow:/
```

3) http://www.xinhuanet.com/robots.txt

新华网的robots.txt文件非常简单,可以看出,它允许所有爬虫抓取该网站的所有页面。实际上,这种做法等同于网站根目录下没有robots.txt文件。

```
# robots.txt for http://www.xinhuanet.com/

User-Agent: *
Allow:/
```

由此可见,Robots协议通过Allow与Disallow的搭配使用,对爬虫的抓取实行限制或放行。如果网站大部分是不允许爬虫抓取的,则可以像淘宝一样,定义允许访问的模式,其他的则不允许。反之,如果大部分是允许抓取的,则可以参考百度的做法,显式定

义不允许访问的模式。结合自己网站的结构，选择合适的编写方法能够简化 robots.txt 文件。例如，某个网站的根目录下有 a1～a10 子目录，只允许爬虫抓取 a10 子目录，按照前面的叙述，写成以下左边的形式就比右边的形式要简洁得多。

```
User-Agent: *
Allow:/a10/
Disallow:/a
```

```
User-Agent: *
Disallow:/a1/
Disallow:/a2/
Disallow:/a3/
Disallow:/a4/
Disallow:/a5/
Disallow:/a6/
Disallow:/a7/
Disallow:/a8/
Disallow:/a9/
```

上述基本功能及书写方式得到了大部分搜索引擎爬虫的支持，除此以外，还有一些拓展功能。虽然它们的普及性还不是很广，但随着行业规范意识的增强，在编写爬虫程序时了解这些拓展功能，对于设计更加友好的爬虫是非常有益的。这些拓展功能主要有通配符的使用、抓取延时、访问时段、抓取频率和 Robots 版本号。

(1) 通配符的使用。Robots 中的许可记录允许使用通配符表示 Disallow 或 Allow 的范围。这些通配符包括 * 、$，需要注意的是问号(?)并不作为一个通配符。通配符 * 代表 0 个或多个任意字符(包括 0 个)。$ 表示行结束符，用来表示至此结束，后面不跟其他任何字符。

通配符举例说明如下。

```
Disallow:/pop/*.html
```

上述语句表示不允许爬虫访问/pop/目录下任何扩展名为.html 的文件。

```
Disallow:/private*/
```

上述语句表示不允许访问以 private 开头的所有子目录。

```
Disallow:/*.asp$
```

上述语句表示不允许访问以.asp 结尾的文件，这就排除了以.aspx 结尾的文件。

(2) 抓取延时。规定爬虫程序两次访问网站的最小时间延时(以秒为单位)，实际上就是规定了爬虫页面抓取的最高请求频率。使用方法如下。

```
Crawl-delay: 10
```

表示两次访问网站的最小时间延时为 10s。

(3) 访问时段。某些网站可能存在业务高峰期，为了使服务器将更多资源分配给它

的用户,不希望爬虫抓取数据。这时可以在许可记录中增加一行:

```
Visit-time: 0100-0500
```

表示允许爬虫在凌晨1:00到5:00访问网站,其他时间段不允许访问。

(4) 抓取频率。这是用来限定URL读取频率的一个选项,使用方法如下。

```
Request-rate: 40/1m 0100 - 0759
```

表示在1:00到07:59之间,以不超过每分钟40次的频率进行访问。

(5) Robots版本号。用来指定Robots协议的版本号,也就是编写robots.txt文件时遵守的Robots协议版本。虽然版本号只是一个选项,但是如果增加这个字段,就会让爬虫程序在解析robots.txt文件时更加方便。例如:

```
Robot-version: Version 2.0
```

指出了所使用的版本为2.0。

目前,上述拓展功能在各网站中的使用还不流行,一个例子是知乎的robots.txt文件(https://www.zhihu.com/robots.txt),其针对EasouSpider的约定就包含了Request-rate、Crawl-delay,也使用了通配符。简书(https://www.jianshu.com/robots.txt)也有类似的使用。

```
User-agent: EasouSpider
Request-rate: 1/2 # load 1 page per 2 seconds
Crawl-delay: 10
Disallow: /login
Disallow: /logout
Disallow: /resetpassword
Disallow: /terms
Disallow: /search
Disallow: /notifications
Disallow: /settings
Disallow: /inbox
Disallow: /admin_inbox
Disallow: /*?guide*
```

扫一扫

视频讲解

3.4 HTTP

超文本传输协议(HTTP)是互联网上广泛使用的一种网络协议,所有WWW文件都必须遵守这个标准。HTTP是基于TCP/IP的应用层协议,采用请求/响应模型。通过使用Web浏览器、网络爬虫或其他工具,客户端向服务器的指定端口(默认端口为80)发送一个HTTP请求,服务器根据接收到的请求向客户端发送响应信息。

3.4.1 HTTP 版本的技术特性

自 1991 年 HTTP 的第一个版本 HTTP 0.9 正式提出以来，HTTP 已经经历了 3 个版本的演化，目前大部分网页使用 HTTP 1.1 作为主要的网络通信协议。

1. HTTP 0.9

HTTP 0.9 是第一个版本的 HTTP，发布于 1991 年。HTTP 0.9 只允许客户端发送 GET 这一种请求，不支持请求头，因此客户端无法向服务器端传递太多消息。

HTTP 0.9 规定服务器只能响应 HTML 格式的字符串，即纯文本，不能响应其他的格式，该协议只用于用户传输 HTML 文档。服务器响应后，TCP 网络连接就会关闭。如果请求的网页不存在，服务器也不会返回任何错误码。

2. HTTP 1.0

1996 年，HTTP 的第二个版本 HTTP 1.0 发布。HTTP 1.0 是以 HTTP 0.9 为基础发展起来的，并且在 HTTP 0.9 的基础上增加了许多新内容。

(1) HTTP 1.0 增加了请求方法。HTTP 1.0 支持 GET、POST、HEAD 等请求方法，每种方法规定的客户端与服务器之间通信的类型不同。

(2) HTTP 1.0 扩大了可处理的数据类型，引入了 MIME(Multipurpose Internet Mail Extensions)机制，除了纯文本外，还可以传输图片、音频、视频等多媒体数据。

(3) HTTP 1.0 在处理 TCP 网络连接时与 HTTP 0.9 相似，但在请求头中加入了 Connection: keep-alive，要求服务器在请求响应后不要关闭 TCP 连接，实现 TCP 连接复用，可以避免 Web 页面资源请求时重新建立 TCP 连接的性能降低。

3. HTTP 1.1

HTTP 1.1 是目前使用最广泛的 HTTP 版本，于 1997 年发布。HTTP 1.1 与之前的版本相比，主要改进集中在提高性能、安全性以及数据类型处理等方面。

(1) HTTP 1.1 与 HTTP 1.0 最大的区别在于默认采用持久连接。客户端不需要在请求头中特别声明(Connection:keep-alive)，但具体实现是否声明依赖于浏览器和 Web 服务器。请求响应结束后 TCP 连接默认不关闭，降低了建立 TCP 连接所需的资源和时间消耗。

(2) HTTP 1.1 支持管道(Pipelining)方式，可以同时发送多个请求。在发送请求的过程中，客户端不需要等待服务器对前一个请求的响应就可以直接发送下一个请求。管道方式提高了请求的传输速度，提高了 HTTP 的效率。但是，服务器在响应时必须按照接收到请求的顺序发送响应，以保证客户端收到正确的信息。

(3) HTTP 1.1 添加了 Host(请求头)字段。随着虚拟主机这种应用架构技术的发展，在一台物理服务器上可以存在多个虚拟主机，这些虚拟主机共享一个 IP 地址。HTTP 1.1 在请求头中加入 Host 字段，指出要访问服务器上的哪个网站。

4. HTTP 2

HTTP 2 于 2015 年发布,在实现与 HTTP 1.1 完全语义兼容的基础上,实现了性能大幅提升,但尚未真正应用。

(1) HTTP 2 允许客户端与服务器同时通过同一个连接发送多重请求/响应消息,实现多路复用。HTTP 2 加入了二进制分帧层,HTTP 消息被分解为独立的帧,帧可以交错发送,然后再根据流标识符在接收端重新拼装,使得 HTTP 2 可以在一个 TCP 连接内同时发送请求和响应。

(2) HTTP 2 压缩大量重复的首部信息,提升通信效率。HTTP 每次通信都会携带首部,当一个客户端想从一个服务器请求许多资源时,可能会出现大量重复的请求头信息。特别是在 HTTP 1.1 中,请求头信息以纯文本的形式发送,大量重复的请求头信息重复传输对网络运输资源的消耗很大。

3.4.2 HTTP 报文

HTTP 报文中存在着多行内容,一般由 ASCII 码串组成,各字段的长度是不确定的。HTTP 报文可分为两种,即请求报文和响应报文。由于目前大部分网站使用 HTTP 1.1 或 HTTP 1.0 版本,所以这里主要介绍这两种报文。

(1) Request Message(请求报文):客户端→服务器端,由客户端向服务器端发出请求,用于向网站请求不同的资源,包括 HTML 文档、图片等。

(2) Response Message(响应报文):服务器端→客户端,服务器响应客户端的请求时发送的回应报文,可以是 HTML 文档的内容,也可以是图片的二进制数据等。

HTTP 1.1 的请求报文和响应报文在形式上与 HTTP 1.0 相同,只是 HTTP 1.1 中添加了一些头部扩充 HTTP 1.0 的功能。因此,本节关于请求报文和响应报文的叙述既适用于 HTTP 1.0,也适用于 HTTP 1.1。

1. 请求报文

HTTP 规定请求报文由起始行、头部(headers)以及实体(entity-body)构成,报文格式如下。

```
<method> <request-URL> <version>
<headers>

<entity-body>
```

第一行是报文的起始行,也称为请求行,由请求方法<method>、请求 URL <request-URL>、协议版本<version>构成。HTTP 1.0 和 HTTP 1.1 支持 GET、POST、HEAD 请求方法,每种方法规定了客户端与服务器之间通信的类型。

接下来是头部<headers>,也称为请求头,包含了客户端处理请求时所需要的信息。根据实际需要,请求头可以是多行的形式。服务器据此获取客户端关于请求的若干配置

参数，如语言种类信息、客户端信息、优先级等内容。头部本质来说是包含若干属性的列表，格式为“属性名：属性值”。如 Accept-Language：zh-CN，将 zh-CN 值赋给 Accept-Language 属性，表示客户端可接收的语言为简体中文。

请求头之后是换行符，表示请求头的结束。

请求头结束标志之后是请求报文的实体< entity-body >，也称为请求体，是传输的内容构成。请求体通过 param1＝value1&param2＝value2 的键值对形式将要传递的请求参数（通常是一个页面表单中的组件值）编码成一个格式化串。

下面是一个向服务器传递 name 和 password 参数的请求体实例。

```
name = Jack&password = 1234
```

值得注意的是，在使用 GET 和 HEAD 方法请求网页时没有请求体。其请求参数表现在请求行，附加在 URL 后面，使用问号（?）来表示 URL 的结尾和请求参数的开始，不同参数之间用“&”连接。例如，下面的命令行表示客户端使用 HTTP 1.0 请求网页 example.html，请求方法为 GET。

```
GET /example.html? name = Jack&password = 1234 HTTP/1.0
User - Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
Accept: text/html
```

这种方式可传递的参数长度受限，并且一些类似银行卡号、密码的敏感内容也不宜出现在 URL 中，解决这些问题的方法就是使用 POST。HEAD 与 GET 方法的区别在于，服务器端接收到 HEAD 请求信息只返回响应头，不会返回响应体；而使用 POST 方法请求时，客户端发送给服务器的参数允许更多，其请求数据都封装在请求体中。

2. 响应报文

与请求报文类似，HTTP 响应报文由起始行、头部（headers）以及实体（entity-body）构成。HTTP 1.0 和 HTTP 1.1 规定的响应报文格式如下。

```
< version >< status >< reason - phrase >
< headers >

< entity - body >
```

响应报文与请求报文的区别在于第一行，响应报文的起始行也称为响应行，由协议版本、状态码、原因短语构成。状态码为表示网页服务器 HTTP 响应状态的 3 位数字，客户端或爬虫可从状态码中得到服务器响应状态，后续将对状态码进行详细介绍。原因短语为状态码提供了文本形式的解释。例如，下面的命令为服务器响应行，以“200 OK”结束，表明响应操作成功。

```
HTTP 1.0 200 OK
```

报文的响应行之后是响应头，包含服务器响应的各种信息，其结构与请求头一致，都

是“属性名:属性值”的格式。

报文的< entity-body >部分是响应体,响应体是 HTTP 要传输的内容。根据响应信息的不同,响应体可以是多种类型的数据,如图片、视频、CSS、JS、HTML 页面或应用程序等。如果客户端请求的是 HTML 页面,则响应体为 HTML 代码。客户端依据响应头中 Content-Type 的属性值对响应体进行解析,如果属性值与响应体不对应,客户端可能无法正常解析。

以下是一个响应报文的例子,表示向客户端成功响应了 text/plain 类型的文本。

```
HTTP/1.0 200 OK
Content - Type: text/plain

< html >
  < body > example </body >
</html >
```

3.4.3 HTTP 头部

在 HTTP 的请求报文和响应报文中都有头部信息块,其中的每个记录具有“属性名:属性值”的形式。例如,请求 http://www.fudan.edu.cn/2016/index.html 页面,从浏览器的开发者模式下观察到的请求头和响应头信息块分别如图 3-10 和图 3-11 所示。除了第一行的请求行和响应行外,其他都是头部信息。

```
GET /2016/index.html HTTP/1.1
Host: www.fudan.edu.cn
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
68.0.3440.106 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
Accept-Language: zh-CN,zh;q=0.9
Cookie: UM_distinctid=163cd68b9db9fe-03062f1baf149c-3c3c5905-100200-163cd68b9de63d
If-None-Match: "5b868fb0-130e1"
If-Modified-Since: Wed, 29 Aug 2018 12:21:04 GMT
```

图 3-10 请求头信息块

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 31 Aug 2018 06:58:38 GMT
Content-Type: text/html
Content-Length: 77969
Last-Modified: Fri, 31 Aug 2018 01:40:41 GMT
ETag: "5b889c99-13091"
Accept-Ranges: bytes
X-Cache: MISS from 2ndDomainSrv
X-Cache-Lookup: MISS from 2ndDomainSrv:80
Via: 1.1 2ndDomainSrv (squid)
Connection: keep-alive
```

图 3-11 响应头信息块

从图 3-10 中可以看出,一些属性在爬虫程序设计中是非常重要的,如 User-Agent、Accept、Accept-Language、Cookie、Content-Type,可以直接在程序中填写相应的属性名和属性值设置爬虫的行为和特征。

属性名可以归纳为三大类,即请求头和响应头都可以使用的属性,以及请求头和响应头分别独有的属性。完整的头部属性名可以在 Wikipedia 网站的 List_of_HTTP_header_fields 页面中查阅(https://en.wikipedia.org/wiki/List_of_HTTP_header_fields),

一些常用的属性名说明如下。

1）Accept

Accept 请求头表示可接受的响应内容。与 Accept 首部类似的还有 Accept-Charset、Accept-Encoding、Accept-Language 等，分别表示客户端可接受的字符集、可接受的编码方式和可接受的语言。

值得注意的是，HTTP 1.1 为这些首部引入了品质因子 q，用来表示不同版本的可用性。例如，在图 3-10 请求头中，Accept 属性值中出现了两个品质因子 q，服务器会优先选取品质因子值高的对应资源版本作为响应。

2）Host

HTTP 1.1 添加 Host 首部表示服务器的域名以及服务器所监听的端口号（如果所请求的端口为所请求服务的标准端口，则端口号可以省略）。例如，图 3-10 请求头中的 Host：www.fudan.edu.cn 字段表示客户端请求访问域名为 www.fudan.edu.cn 的服务器的 80 端口。HTTP 1.0 认为每台服务器都绑定一个唯一的 IP 地址，但目前多个虚拟站点可以共享同一个 IP 地址。在加入 Host 首部后，其属性值可以明确指出要访问 IP 地址上的哪个站点。应当指出的是，请求头中必须包含 Host 首部，否则服务器会返回一个错误。

3）Range

HTTP 1.1 在请求头中添加 Range 首部表示客户端向服务器请求资源的某个部分。例如，Range：bytes＝0-499 表示客户端请求资源的前 500 字节。Range 首部的使用避免了服务器向客户端发送其不需要的资源。从 Web 服务器下载文件时，所使用的断点续传功能就依赖于这个首部的使用。

4）User-Agent

User-Agent 首部表示客户端的身份标识字符串。通过该字符串使得服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等信息。对于浏览器，该字符串的标准格式为

```
浏览器标识（操作系统标识；加密等级标识；浏览器语言）渲染引擎标识 版本信息
```

一个例子如下。

```
User - Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/68.0.3440.106 Safari/537.36
```

对于爬虫程序，该属性的值是可以随便设置的，并不一定要遵守浏览器和格式，但是这种方式增加了服务器识别爬虫的风险。

5）Content-Range

与请求头的 Range 属性相对应，HTTP 1.1 在响应头中添加 Content-Range 首部表示服务器已响应客户端请求的部分资源。例如，Content-Range：bytes 0-499/1024 表示已经响应了实体的前 500 字节，斜杠后面的数字表示实体的大小。

6) Content-Type

Content-Type 首部表示响应体的 MIME 类型,即响应体是用何种方式编码的。常见的媒体编码格式类型有 text/html(HTML 格式)、text/plain(纯文本格式)、image/jpeg(JPG 图片格式);以 application 开头的媒体格式类型有 application/json(JSON 数据格式)、application/pdf(PDF 格式)、application/msword(Word 文档格式)。

7) Cookie

Cookie 是请求报文中可用的属性,也是客户端最重要的请求头属性。Cookie 存储了客户端的一些重要信息,如身份标识、所在地区等,通常是一个文本文件。在向服务器发送 URL 请求时可以将文件内容读出,附加在 HTTP 的请求头中,能免去用户输入信息的麻烦。

8) Set-Cookie

Set-Cookie 是响应报文中可用的属性,服务器可以在响应报文中使用该属性将一些信息推送给客户端。客户端收到信息后,通常的做法是生成 Cookie 文件,将这些内容保存起来。例如,以下代码表示服务器发送 UserID、Version 信息给客户端,希望客户端将它们保存到 Cookie 文件中。

```
Set - Cookie: UserID = Wang12; Version = 1
```

9) Connection

Connection 是请求报文和响应报文中都可用的属性,通过该属性可以允许客户端和服务器指定与请求/响应连接有关的选项,相应的属性值有 keep-alive、Upgrade。

从前面关于 HTTP 版本技术特性演变的介绍可以看出,keep-alive 是一个很重要的属性。然而,并不是设置了 keep-alive 就意味着可以建立永久连接。在默认情况下,Web 服务端设置了 keep-alive 的超时时间,当连接超过指定的时间时,服务端就会主动关闭连接。此外,为了缩短 HTTP 响应时间,避免 Web 服务拥塞,并发连接数(即同时处于 keep-alive 的连接的数目)也不宜太大。

10) Content-Length

Content-Length 首部表示响应体的长度,其属性值用八进制字节表示。

11) Content-Language

Content-Language 首部表示响应内容所使用的语言。

12) Server

Server 首部表示服务器名称。

13) Warning

HTTP 1.1 在响应头中添加 Warning 首部以更好地描述错误和警告信息,这些信息通常比原因短语更详细。

14) Referer

HTTP Referer 是 header 的一部分,当浏览器向 Web 服务器发送请求时一般会带上 Referer 属性值,告诉服务器该请求是从哪个页面链接过来的。例如,B 页面上有一个链

接到 A 页面，当用户从该链接访问 A 页面时，发送给 A 页面所在的 Web 服务器的请求头中的 Referer 值就是 B。

3.4.4　HTTP 状态码

HTTP 状态码（HTTP Status Code）是用来表示 Web 服务器 HTTP 响应状态的 3 位数字代码。通过 HTTP 状态码可以得知服务器的响应状态，以便更好地处理通信过程中遇到的问题。对于爬虫程序，可以通过这个状态码确定页面抓取结果。HTTP 状态码由 RFC 2616 规范定义，并得到 RFC 2518、RFC 2817、RFC 2295、RFC 2774、RFC 4918 等规范扩展。状态码有 5 种类别，即信息、成功、重定向、请求错误和服务器错误。

状态码由 3 位数字构成，如 200 表示请求成功。数字中的第一位代表了状态码所属的响应类别，共有五大类状态码，分别以数字 1～5 开头，如表 3-1 所示。

表 3-1　状态码的类别

状态码	类　　别	分 类 描 述
1××	信息状态码	表示服务器已经收到请求，需要继续处理。在 HTTP 1.0 中没有定义 1××状态码，因此除非在某些试验条件下，服务器不要向客户端发送 1××响应
2××	成功状态码	表示请求被成功接收并处理
3××	重定向状态码	表示需要采取进一步操作才能完成请求。这类状态码用来重定向，对于 GET 或 HEAD 请求，服务器响应时会自动将客户端转到新位置
4××	客户端错误状态码	表示客户端的请求可能出错，服务器无法处理请求
5××	服务器错误状态码	表示服务器在处理请求的过程中内部发生了错误

服务器返回的状态码后常跟有原因短语（如 200 OK），原因短语为状态码提供了文本形式的解释。

状态码的个数很多，被 RFC 2616、RFC 2518、RFC 2817 等规范定义的状态码有 60 多个，但实际常见的状态码仅有 10 个，通常用在爬虫程序中。

1）成功状态码

200（OK）是最常见的状态码，表示服务器成功处理了请求，同时请求所希望的响应头或实体随着响应返回。

202（Accepted）表示服务器已接受请求，但尚未处理。

2）重定向状态码

301（Moved Permanently）表示请求的资源已经永久地移动到新位置，即被分配了新的 URL。客户端以后的新请求都应使用新的 URL。

304（Not Modified）表示客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容并没有改变。在返回的响应报文中不含实体的主体部分。

3）客户端错误状态码

400（Bad Request）表示请求报文存在语法错误或参数错误，服务器无法理解。

401(Unauthorized)表示当前请求要求客户端进行身份认证或是身份认证失败。对于需要登录的网页,服务器可能返回此响应。

403(Forbidden)表示服务器已经理解请求,但是拒绝执行。拒绝执行可能是客户端无访问权限等原因,但服务器无须给出拒绝执行的理由。

404(Not Found)表示请求失败,在服务器上无法找到请求的资源。

4) 服务器错误状态码

500(Internal Server Error)表示服务器执行请求时发生错误,无法完成请求。当服务器端的源代码出现错误时会返回这个状态码。

503(Server Unavailable)表示服务器超负载或正停机维护,无法处理请求。这个状态通常是临时的,将在一段时间后恢复。

3.4.5 HTTPS

超文本传输安全协议(Hypertext Transfer Protocol Secure,HTTPS)是一种通过计算机网络进行安全通信的传输协议,简单来说,HTTPS 是 HTTP 的安全版本。HTTP 报文使用明文发送,传输的信息很容易被监听和篡改;HTTPS 是使用 SSL/TLS 加密的 HTTP,可以保护传输数据的隐私和完整性,同时实现服务器的身份验证。

SSL(安全套接层)是介于 HTTP 与 TCP 之间的安全协议,SSL 在传输层同时使用对称加密以及非对称加密对网络连接进行加密。其中,对称加密算法的加密与解密使用同一个密钥;非对称加密算法的加密与解密使用不同的两个密钥,即公开密钥(Public Key,简称公钥)和私有密钥(Private Key,简称私钥)。非对称加密与对称加密相比安全性更高,但是加密与解密花费的时间长、速度慢。因此,在准备建立连接时,SSL 使用服务器的证书(公钥)将对称密钥非对称加密,保证对称密钥的安全;在连接建立后,SSL 对数据量较大的传输内容使用对称加密,提高加密效率。

传输层安全协议(Transport Layer Security Protocol,TLS)是 SSL 3.0 的后续版本,与 SSL 的内容大致相同,因此很多文章将 TLS 与 SSL 并列称呼,即 SSL/TLS。

HTTPS 相当于在 HTTP 的基础上使用 SSL/TLS 对传输的数据进行加密,因此 HTTPS 与 HTTP 交互最大的区别在于,使用 HTTPS 传输数据之前需要客户端与服务器进行一次 SSL 握手,在握手过程中将确立双方加密传输数据的密码信息。如图 3-12 所示,下面主要介绍 SSL 握手中的单向认证过程,即客户端校验服务器的证书合法性。

(1) 客户端的浏览器发送客户端 SSL 协议的版本号、加密算法的种类、产生的随机数以及其他在 SSL 协议中需要用到的信息。

(2) 服务器向客户端返回 SSL 协议的版本号、加密算法的种类、随机数以及其他相关信息,同时服务器还将向客户端发送自己的证书。证书用于身份验证,其中包含用于非对称加密的公共密钥。

(3) 客户端用服务器传过来的证书进行服务器信息校验。服务器信息校验的内容包括证书链是否可靠、证书是否过期、证书域名是否与当前的访问域名匹配等。如果服务器信息校验没有通过,则结束本次通信;否则继续进行步骤(4)。

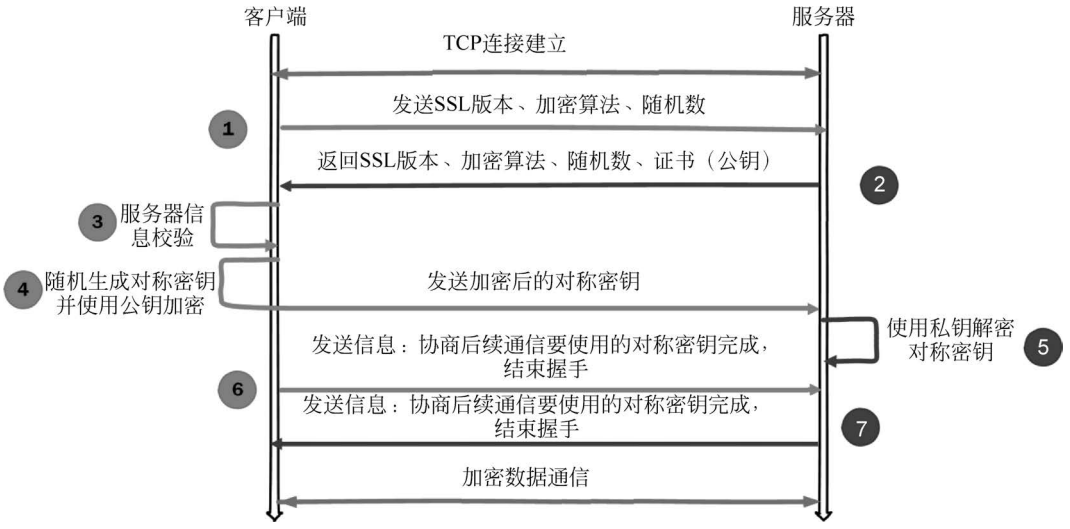

图 3-12　HTTPS 交互过程

(4) 客户端随机生成一个用于后续通信的对称密钥，然后使用服务器在步骤(2)发来的公共密钥对其加密，之后将加密的对称密钥发送给服务器。

(5) 服务器使用私有密钥解密客户端发来的信息，得到用于后续通信的对称密钥。

(6) 客户端向服务器发送信息，信息中指明协商后续通信要使用的对称密钥完成，后面的通信都要使用步骤(5)得出的对称密钥进行加密，同时通知服务器握手过程结束。

(7) 服务器也向客户端发送协商后续通信要使用的对称密钥完成，握手过程结束。

之后 SSL 握手结束，客户端和服务器开始使用相同的对称密钥对要传输的数据进行加密，同时进行通信完整性的检验。

3.5　状态保持技术

当使用浏览器访问 Web 服务器上的页面时，浏览器首先会建立与 Web 服务器的 HTTP 连接，之后浏览器在这个连接上发送 URL，接收服务器返回的信息，并显示在浏览器上。如果进行连接复用或持久化，后续的 URL 请求和响应信息接收都可以在这个连接上进行，这些请求是相互独立的。

如果前后两次 URL 请求之间需要共享某些数据，如在第一个页面进行了用户登录，第二个页面的访问就需要以该用户身份进行。更简单地讲，一个客户端访问服务器时，可能会在这个 Web 服务器上的多个页面之间不断刷新、反复连接同一个页面或向一个页面提交信息。在这种情况下，不同页面之间或同一个页面的不同次访问之间需要保持某种状态，实现这种需求的技术就是状态保持技术。

由于 HTTP 本身是无状态的，客户端向服务器发送一个请求，然后服务器返回一个响应，不同 URL 之间的状态无法共享，即状态无法保持。在一个通信系统中要实现状态保持，只能从客户端和服务器两个角度来设计，相应地，这两种状态保持技术就是 Cookie 和 Session。

3.5.1 Cookie

Cookie 是由服务器生成,并在客户端进行保存和读取的一种信息,Cookie 通常以文件形式保存在客户端。查看 Cookie 的方法因浏览器的不同而不同,图 3-13、图 3-14 所示分别为在 Chrome 和 Edge 浏览器中查看 Cookie 的方法。

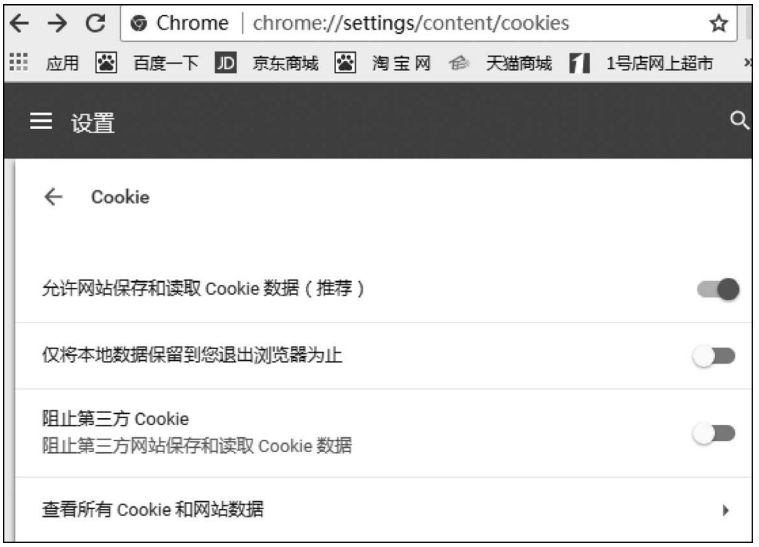

图 3-13 在 Chrome 浏览器中查看 Cookie 的方法

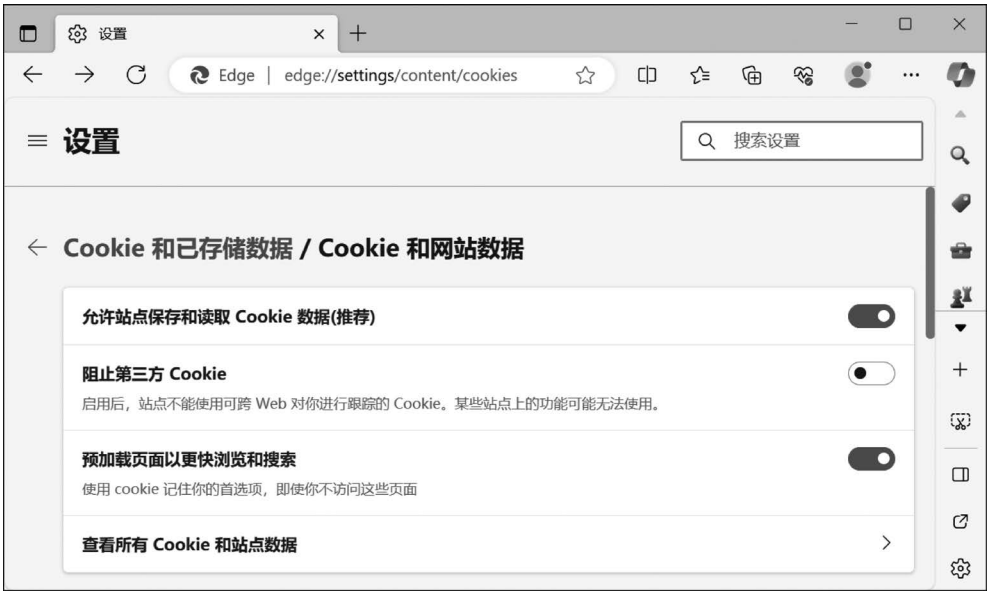

图 3-14 在 Edge 浏览器中查看 Cookie 的方法

每个 Cookie 记录中包含了名字、值、过期时间、路径和域,具体解释如表 3-2 所示。

表 3-2　Cookie 记录内容解释

一个 Cookie 记录	对应行的解释	取 值 样 例
name	Cookie 变量名	uid
value	Cookie 变量值	abcxyz001
domain＋path	Cookie 变量所属的域、路径，即作用范围	mail. fudan. edu. cn/
option	可选标志	1600
a	过期时间(FILETIME 格式)的高位整数	1638219776
b	过期时间(FILETIME 格式)的低位整数	36669619
c	创建时间(FILETIME 格式)的高位整数	1256217104
d	创建时间(FILETIME 格式)的低位整数	30651615
*	Cookie 记录分隔符	

变量名(name)和变量值(value)并没有特殊的命名规定，也没有写法上的限制。由于 Cookie 是在服务器生成，最终由浏览器进行解析，所以这些名称和值一般与页面的某些变量有关。

域(domain)是指 Cookie 变量所属的域，也就是该 Cookie 的有效域范围。路径(path)是接在域后面的 URL 路径，最简单的路径是/，也可以是/image 等形式。路径与域合在一起就构成了 Cookie 的作用范围，表示 Cookie 变量在该范围内的网页都有效，不管用户访问该目录中的哪个网页，浏览器都会将该 Cookie 信息附在网页头部中，并发送给服务器。

如果设置了过期时间，浏览器会把 Cookie 保存到硬盘上，当关闭后再次打开浏览器时，这些 Cookie 仍然有效直到超过设定的过期时间。如果没有设置过期时间，则表示这个 Cookie 的生命期为浏览器会话期，只要关闭浏览器窗口，Cookie 就消失了。这种生命期为浏览器会话期的 Cookie 称为会话 Cookie。会话 Cookie 一般不存储在硬盘上，而是保存在内存里。

这里举一个例子，如图 3-15 所示，在一个提供页面登录的 Web 页面(http://mail. fudan. edu. cn/)中勾选“记住用户名”选项，希望浏览器记住用户名，之后随便输入一个用户名，如 abcxyz001，单击“登录”按钮，再重新加载该页面，就会发现浏览器自动填充了用户名。

图 3-15　某 Web 登录页面

这时查看 mail. fudan. edu. cn 的 Cookie 文件就会发现，文件中的一个记录如图 3-16 所示，可以看出 name 和 value 分别为 uid、abcxyz001。uid 实际上就是图 3-15 中用户名输入框的名字。

在了解了 Cookie 的内容之后，需要了解 Cookie 的创建和使用方法。Cookie 中记录的内容是在服务器生成的，然后通过 HTTP headers 从服务器发送到浏览器上。

浏览器在处理用户输入的 URL 时，根据其中的域名寻找是否有相应的 Cookie，如果

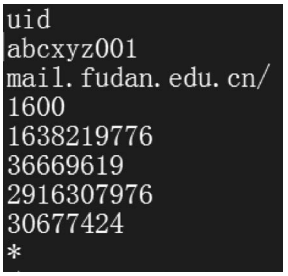

图 3-16 Cookie 文件中的一个记录

有,则进一步检查其中的作用范围。当作用范围大于或等于 URL 指向的位置时,浏览器会将该 Cookie 中的内容读出,附在请求资源的 HTTP 请求头上并发送给服务器。

具体的创建和使用方式与 Web 服务的框架和支持的语言有一定关系,但基本原理是一样的。以上面的登录处理为例,这是一个 JSP 页面,假如为 Login.jsp,单击“登录”按钮之后,将输入的用户名、密码提交给 action="results.jsp"进行处理。那么在 results.jsp 中,下面的脚本将用户输入的用户名作为 Cookie 的值(value),其对应的名称(name)为 username。最后通过响应让浏览器生成一个 Cookie。

```
String username = request.getParameter("username");
Cookie cookie1 = new Cookie("username",username);   //创建 Cookie
cookie1.setMaxAge(3600);                            //设置过期时间,以秒为单位
cookie1.setPath("/");                               //设置路径
response.addCookie(cookie1);                        //发送到浏览器执行创建动作
```

当再次访问这个 JSP 页面时,浏览器自动读取 Cookie 内容并附加在请求头中,这样服务器就可以通过下面的代码获得浏览器发送的 Cookie 内容,并从中读取出用户名。这段代码可以嵌入自动输入用户名的页面中。

```
Cookie cookies[] = request.getCookies();
Cookie sCookie = cookies[1];
String username = sCookie.getValue();
```

接着就可以按照下面的方式将 username 变量赋值给输入框,然后发送给浏览器,这样在浏览器上就能看到填充的用户名了。

```
<input type = "text" name = "username" value = <% = username %> size = "44">
```

扫一扫

视频讲解

3.5.2 Session

仅使用 Cookie 保持状态存在一定的问题,主要表现在以下几方面。

(1) Cookie 虽然可以灵活地增加 name 和 value,但每个域名的 Cookie 数量、Cookie 文件大小是受限的,具体数值取决于不同浏览器。例如,Firefox 将每个域名的 Cookie 限制为最多 50 个,每个文件最大 4KB。因此,在一些需要在客户端和 Web 服务器之间进行较多数据交换的应用中,使用 Cookie 就不合适。

(2) Cookie 存在一定的安全风险,Cookie 信息可以很容易被截获,如果是明文,则可能造成信息泄露。但不管是明文还是密文,都可以直接被用来伪造 HTTP 请求,即 Cookie 欺骗,冒充受害人的身份登录网站。参考文献[8]展示了一个实际的 Cookie 被窃取的案例。

Session 是另一种常见的在客户端与服务器之间保持状态的机制，在一定程度上解决或缓解了上述问题，准确理解其技术原理有利于设计更好的动态爬虫。

Session 可以看作 Web 服务器上的一个内存块，能够将原本保存在 Cookie 中的用户信息存储在该内存块中，而客户端和服务器之间依靠一个全局唯一标识 Session_id 访问 Session 中的用户数据，这样只需要在 Cookie 中保存 Session_id 就可以实现不同页面之间的数据共享。可见，在 Session 机制下，除了 Session_id 外，其他用户信息并不保存到 Cookie 文件中，这解决了上述两个问题。

虽然 Session_id 也可能会像 Cookie 内容一样被截获，但通常 Session_id 是加密存储的。对于安全性保障更有效的机制是它的动态性，即 Session 是在调用 HttpServletRequest.getSession(true)这样的语句时才被创建，而在符合下面 3 个条件之一时终止。

(1) 程序调用 HttpSession.invalidate()。

(2) 服务器关闭或服务停止。

(3) Session 超时，即在一定的连续时间内服务器没有收到该 Session 所对应客户端的请求，并且这个时间超过了服务器设置的 Session 超时的最大时间。这个最大时长一般是 30min，在服务器上可配置。

因此，只要 Session 在内存中的存活时间不是太长，当攻击者截获某个 Session_id 时，其对应的内存 Session 可能已经销毁了，也就无法获得用户数据。

根据以上描述，可以得出 Session、Cookie、服务器和客户端(浏览器、爬虫)之间的关系，如图 3-17 所示。

图 3-17　Session、Cookie、服务器和客户端的关系

除了提升安全性外，Session 由于保留了客户端请求的内存块，因此当该请求的处理需要连接数据库等额外操作时，使用 Session 会大大提升请求效率。在 12.3 节的例子中展示了这种优势。

思考题

1. Web 服务器及应用架构与网络爬虫之间是什么关系？
2. 使用 Tomcat 等 Web 服务器软件制作页面，并在浏览器中浏览。
3. 浏览器需要遵守 Robots 协议吗？Robots 协议中主要规定了哪些方面的内容？
4. 不同版本的 HTTP 的主要区别是什么？
5. 使用浏览器的开发者工具查看 HTTP 请求报文、响应报文、状态码和 Cookie。
6. 谈谈 Cookie 的作用。