第 3 章 网页抓取技术

在搜索引擎系统中，网页抓取技术是整个系统的基础。网络爬虫根据 HTTP，向网站服务器发送请求报文，网站服务器接收并分析请求，向网络爬虫返回网页报文。本章详细讲解网络爬虫的基本原理、抓取策略和关键技术。

3.1 搜索引擎爬虫

网络爬虫(Crawler)，也称为蜘蛛程序(Spider)，或网络机器人(Robots)。网络爬虫是一个自动提取网页的程序，是搜索引擎的重要组成部分。作为爬虫，就是尽可能多和快地给索引部分输送网页，实现强大的数据支持。网络爬虫是通过网页的链接地址来寻找网页，从网站某一个页面(通常是首页)开始，读取网页的内容，找到在网页中的其他链接地址，然后通过这些链接地址寻找下一个网页，这样一直循环下去，直到把这个网站所有的网页都抓取完为止。如果把整个互联网当成一个网站，那么网络爬虫就可以用这个原理把互联网上所有的网页都抓取下来。

3.1.1 网络爬虫工作原理

在互联网中，网页之间的链接关系是无规律的，它们的关系非常复杂。如果一个爬虫从一个起点开始爬行，那么它将会遇到无数多的分支，由此生成无数条的爬行路径，如果任意爬行，就有可能永远也爬不到头，因此要对它加以控制，制定其爬行的规则。世界上没有一种爬虫能够抓取全互联网所有的网页，所以就要在提高其爬行速度的同时，提高其爬行网页的质量。

网络爬虫在搜索引擎中占有重要位置，对搜索引擎的查全、查准都有影响，决定了搜索引擎数据容量的大小，而且网络爬虫的好坏直接影响搜索结果页中的死链接(即链接所指向的网页已经不存在)的个数。搜索引擎爬虫有深度优先策略和广度优先策略。另外，识别垃圾网页、避免抓取重复网页，也是高性能爬虫的设计目标。

爬虫的作用是为搜索引擎抓取大量的数据，抓取的对象是整个互联网上的网页。爬虫程序不可能抓取所有的网页，因为在抓取的同时，Web 的规模也在增大，所以一个好的爬虫程序一般能够在短时间内抓取更多的网页。一般爬虫程序的起始点都选择在一个大型综合性的网站，这样的网站已经涵盖了大部分高质量的站点，爬虫程序就沿着这些链接爬行。在爬行过程中，最重要的就是判断一个网页是否已经被爬行过。爬虫的运行过程如图 3-1 所示。

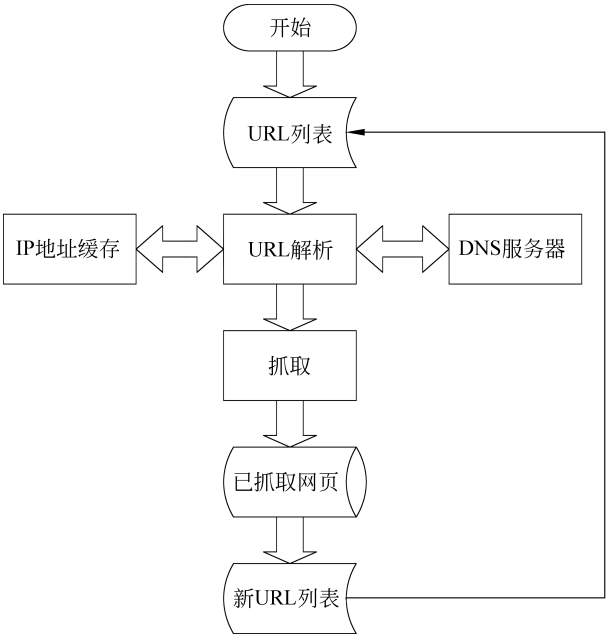

图 3-1　爬虫的运行过程

在爬虫开始的时候，需要给爬虫输送一个 URL 列表，这个列表中的 URL 地址便是爬虫的起始位置，爬虫从这些 URL 出发，开始爬行，一直不断地发现新的 URL，然后再根据策略爬行这些新发现的 URL，如此永远反复下去。一般的爬虫都自己建立 DNS 缓冲，建立 DNS 缓冲的目的是加快 URL 解析成 IP 地址的速度。

Google 为了获取上亿的网页，设计了一个分布式的爬行系统。一个 URL 服务器将 URL 列表提供给网络爬行器(Google 同时运行 3 个爬行器)。每个爬行器同时保持大约 300 个网络连接。在最高速度的时候，通过 4 个爬行器，该系统可以每秒钟获取超过 100 个网页。影响爬行速度的一个重要因素是 DNS 查询，为此，每个爬行器都要维护一个自己的 DNS 缓冲。这样每个连接都处于不同的状态，包括：DNS 查询、连到主机、发送请求、得到响应。这些因素综合起来使得爬行器变成一个非常复杂的系统。它通过异步输入/输出来管理事件，通过一定数量的队列来管理获取网页过程中的状态迁移。

3.1.2　开源网络爬虫简介

1. Heritrix 爬虫

Heritrix 工程始于 2003 年年初，最初的目的是开发一个特殊的爬虫，对网上的资源进行归档，建立网络数字图书馆。Heritrix 的最出色之处在于它的可扩展性，开发者可以随意地扩展它的各个组件，来实现自己的抓取逻辑。缺点是单实例的爬虫，之间不能进行合作，配置比较麻烦。下载网址为 https://sourceforge.net/projects/archive-crawler/。

爬虫的执行过程是递归进行的，主要有以下几步。

(1) 在预定的 URL 中选择一个。

(2) 获取 URL。

(3) 分析,归档结果。

(4) 选择已经发现的感兴趣的 URL,加入预定队列。

(5) 标记已经处理过的 URL。

Heritrix 主要有 3 大部件:范围部件、边界部件和处理器链。

(1) 范围部件:主要按照规则决定将哪个 URL 入队。

(2) 边界部件:跟踪哪个预定的 URL 将被收集,以及已经被收集的 URL,选择下一个 URL,剔除已经处理过的 URL。

(3) 处理器链:包含若干处理器获取 URL,分析结果,将它们传回给边界部件。

2. WebLech 爬虫

WebLech 是一个功能强大的 Web 站点下载与镜像工具,它支持按功能需求来下载 Web 站点并且能够尽可能模仿标准 Web 浏览器的行为。WebLech 是多线程的,提供了一个 GUI 控制台。下载网址为 http://weblech.sourceforge.net/。

WebLech 爬虫的工作过程如下。

(1) 抓取网页。通过对某一网站主页进行分析,选择其上面的所有链接,然后把这些网址加入到要下载的队列中。

(2) 加入队列。加入到队列时,可以广度优先,也可以深度优先。一般情况下,是广度优先,一般搜索引擎只抓取网站的前三层网页。通常超过三层的网页,网页质量一般不高。

(3) 多线程技术。爬虫是很注重效率的,所以一般采取多线程或多进程技术,WebLech 的做法是,多线程访问维护下载队列,把所有要下载的网址加入到队列中去,每一个线程依次从队列中取出网址进行下载,当然队列的访问是互斥的。

(4) 程序中断处理。一般情况下网络爬虫会因为网络的不确定性而经常中断,WebLech 的做法是设置一个时间间隔,之后对队列状态进行维护,然后进行下载。

(5) 防止对同一网址多次下载。一般会把所下载过的网址写入 Hash 表,从而避免下载同一网页。

(6) 像一些著名搜索引擎,爬虫肯定都是分布式的,但 WebLech 不是分布式的,所以 WebLech 相对简单些。

(7) 过滤网址。WebLech 可以设 urlMatch 的值对网址进行过滤,也就是说,网址中必须出现子串 urlMatch,这样可以更好地定制网址。

3. JSpider 爬虫

JSpider 是一个完全可配置和定制的 Web Spider 引擎,可以利用它来检查网站的错误,检查网站内外部链接,分析网站的结构(可创建一个网站地图),下载整个 Web 站点。JSpider 的行为是由配置文件具体配置的,比如采用什么插件,结果存储方式等都在 conf\[ConfigName]\目录下设置。JSpider 默认的配置种类很少,用途也不大。但是 JSpider 非常容易扩展,可以利用它开发强大的网页抓取与数据分析工具。要做到这些,需要对 JSpider 的原理有深入的了解,然后根据自己的需求开发插件,撰写配置文件。其下载网址为 http://j-spider.sourceforge.net/。

JSpider 的运行非常简单：

(1) 首先下载 jspider-0.5.0-dev.zip，然后解压缩。

(2) 单击“开始”→“运行”→cmd，进入命令行窗口，然后进入 jspider-0.5.0-dev/bin 目录。

(3) 例如，要抓取网站 http：//www.bjfu.edu.cn 的内容，可在命令窗口中输入命令：

```
jspider http://www.bjfu.edu.cn download
```

(4) 运行一会儿后，关闭窗口，到根目录下的 output 文件夹中就可以看到抓取到的网页了。

4. WebSPHINX 爬虫

WebSPHINX 是一个 Java 类包和 Web 爬虫的交互式开发环境，主要由工作平台和类库构成。它能根据用户提供的 URL，自动提取网页中的链接。并且根据判断出的链接自动实现在 Internet 中的漫游。下载网址为：http://www.cs.cmu.edu/~rcm/websphinx/。

WebSPHINX 分为简单模式和高级模式。在简单模式中可设置 Robot 在一个服务器内或在整个 Internet 中的漫游，采取超链接加图片链接访问方式，具有可选性。此外，还可设置访问是广度优先还是深度优先。高级模式除了具备简单模式的所有功能外，还能设置规则，如 HTML 标签、XML 标签、脚本语言、锚等，从而实现更细化的查询。高级模式中还可以设置访问网络的线程数、访问超时、页面大小、保存、提取等动作。

WebSPHINX 遍历整个网络时，用一个哈希表来存放它所访问过的 URL，并用 URL 作为哈希表中的关键字，这样就不会因重复访问同一个 URL 而使搜索程序陷入死循环。在访问过程中，它还可以检查 URL 在网络中的位置，并根据该位置来设定它的优先级数。

5. Arachnid 爬虫

Arachnid 是一个基于 Java 的 Web Spider 框架。它包含一个简单的 HTML 剖析器，能够分析包含 HTML 内容的输入流。通过实现 Arachnid 的子类就能够开发一个简单的 Web Spiders，并能够在 Web 站上的每个页面被解析之后增加几行代码调用。Arachnid 的下载包中包含两个 Spider 应用程序例子用于演示如何使用该框架。

下载网址为 http://arachnid.sourceforge.net/。

3.1.3 网页信息的抓取

搜索引擎建立网页索引，处理的对象是文本文件。对于网络爬虫来说，抓取下来的网页包括各种格式，如 HTML、图片、DOC、PDF、多媒体、动态网页及其他格式等。这些文件抓取下来后，需要把这些文件中的文本信息提取出来。准确提取这些文档的信息，一方面对搜索引擎的搜索准确性有重要作用，另一方面对于网络爬虫正确跟踪其他链接有一定影响。

1. 普通网页信息的抓取

对于 DOC、PDF 等文档，这种由专业厂商提供的软件生成的文档，厂商都会提供相应

的文本提取接口。搜索引擎爬虫只需要调用这些插件的接口，就可以轻松地提取文档中的文本信息和文件其他相关的信息。

HTML 等文档不一样，HTML 有一套自己的语法，通过不同的命令标识符来表示不同的字体、颜色、位置版式等，提取文本信息时需要把这些标识符都过滤掉。过滤标识符并非难事，因为这些标识符都有一定的规则，只要按照不同的标识符取得相应的信息即可。但在识别这些信息的时候，需要同步记录许多版式信息，例如，文字的字体大小、是否是标题、是否是加粗显示、是否是页面的关键词等，这些信息有助于计算单词在网页中的重要程度。同时，对于 HTML 网页来说，除了标题和正文以外，会有许多广告链接以及公共的频道链接，这些链接和文本正文一点儿关系也没有，在提取网页内容的时候，也需要过滤这些无用的链接。例如，某个网站有“产品介绍”频道，因为导航条在网站内每个网页都有，若不过滤导航条链接，在搜索“产品介绍”的时候，则网站内每个网页都会搜索到，无疑会带来大量垃圾信息。过滤这些无效链接需要统计大量的网页结构规律，抽取一些共性，统一过滤；对于一些重要而结果特殊的网站，还需要个别处理。这就需要搜索引擎爬虫的设计有一定的扩展性。

对于多媒体、图片等文件，一般是通过链接的锚文本（即链接文本）和相关的文件注释来判断这些文件的内容。例如，有一个链接文字为“故宫的照片”，其链接指向一张 bmp 格式的图片，那么搜索引擎爬虫就知道这张图片的内容是“故宫的照片”。这样，在搜索“故宫”和“照片”的时候都能让搜索引擎找到这张图片。另外，许多多媒体文件中有文件属性，考虑这些属性也可以更好地了解文件的内容。

2. 动态网页信息的抓取

动态网页一直是网络蜘蛛面临的难题。动态网页是相对于静态网页而言的，是由程序自动生成的页面，这样的好处是可以快速统一更改网页风格，也可以减少网页所占服务器的空间，但这样给网络爬虫的抓取带来一些麻烦。由于开发语言不断增多，动态网页的类型也越来越多，如 asp、jsp、php 等。这些类型的网页对于网络爬虫来说，可能还稍微容易一些。网络爬虫比较难于处理的是一些脚本语言（如 VBScript 和 JavaScript）生成的网页，如果要完善地处理好这些网页，网络爬虫需要有自己的脚本解释程序。对于许多数据是放在数据库中的网站，需要通过本网站的数据库搜索才能获得信息，这些给网络爬虫的抓取带来很大的困难。对于这类网站，如果网站设计者希望这些数据能被搜索引擎搜索，则需要提供一种可以遍历整个数据库内容的方法。

网页内容的提取，一直是网络爬虫中重要的技术。整个系统一般采用插件的形式，通过一个插件管理服务程序，遇到不同格式的网页采用不同的插件处理。这种方式的好处在于扩充性好，以后每发现一种新的类型，就可以把其处理方式制作成一个插件补充到插件管理服务程序之中。

3. 爬虫更新周期

由于网站的内容经常在变化，因此网络爬虫也需不断地更新其抓取网页的内容，这就需要网络爬虫按照一定的周期去扫描网站，查看哪些页面是需要更新的页面，哪些页面是新增页面，哪些页面是已经过期的死链接。

搜索引擎的更新周期对搜索引擎搜索的查全率有很大影响。如果更新周期太长，则

总会有一部分新生成的网页搜索不到；如果周期过短，技术实现会有一定难度，而且会对带宽、服务器的资源都有浪费。网络爬虫并不是所有的网站都采用同一个周期进行更新，对于一些重要的更新量大的网站，更新的周期短，如有些新闻网站，几个小时就更新一次；相反对于一些不重要的网站，更新的周期就长，可能一两个月才更新一次。

一般来说，网络爬虫在更新网站内容的时候，不用把网站网页重新抓取一遍，对于大部分的网页，只需要判断网页的属性（主要是日期），把得到的属性和上次抓取的属性相比较，如果一样则不用更新。

4. Ajax 网站内容的抓取

随着以 Ajax（Asynchronous JavaScript and XML）驱动为代表的 Web 2.0 应用的大规模流行，如何抓取 Ajax 网站的内容成了目前搜索引擎爬虫的巨大难题。

现有的网络爬虫只需模拟用户的超级链接请求并解析对应的响应页面，然后再分析页面内容、语义以及衍生的超级链接，以此进行网页内容的抓取。Ajax 与以往应用的最大不同之处在于将超级链接驱动的请求/响应更多地改用了 JavaScript 驱动的异步请求/响应。而对于现有的网络爬虫，一般来说，它们对 JavaScript 是缺乏语义理解的，它们很难再去模拟触发 JavaScript 的异步调用并解析返回的异步回调逻辑和内容。

Ajax 应用打破了以往基于纯 HTTP 请求/响应协议的网络爬虫机制，即默认所有的页面资源都是直接由超级链接所触发并指向的。对于传统的 Web 应用网络爬虫来说，它们默认每个页面的 DOM 结构是相对静态不变的。对于用户的操作，JavaScript 会对 DOM 结构进行大量的变动，甚至页面所有的内容都是通过 JavaScript 直接从服务器端读取并动态绘制出来的。

传统的网络爬虫引擎大都是协议驱动的，而面对抓取 Ajax 的爬虫引擎需要的是事件驱动。这样，在新的引擎中，要做到事件驱动，需要考虑以下 3 大关键问题。

（1）JavaScript 的交互分析和解释。

（2）DOM 事件的处理和解释分发。

（3）动态 DOM 内容语义的抽取。

在新一代的 Web 应用中，这个网络爬虫机制的问题除了在 Ajax 中存在之外，在其他的 Flash/Flex 以及 WPF/E、XUL 应用等中也是普遍存在的。而且对于 Flash/Flex 来说，问题可能更为严重，因为所有的 ActionScript 代码是编译执行的，这在效率提高的同时，又无疑带给了网络爬虫巨大的困难。

3.2 搜索引擎爬虫的关键技术

网络爬虫是搜索引擎的关键部分。爬虫从互联网上获取消息，然后交给搜索引擎的其他部分进行处理，最后返回给用户。下面讨论爬虫在工作时采取哪些技术和算法，以及用何种策略进行网页的抓取。

3.2.1 网页抓取优先策略

网页抓取优先策略也称为“页面选择问题”（Page Selection），通常是尽可能地首先抓

取重要的网页，这样保证在有限的资源内尽可能地照顾到那些重要性高的网页。重要性度量由链接欢迎度、链接重要度和平均链接深度这3个方面决定。

1. 链接欢迎度

链接欢迎度IB(P)主要由反向链接(Backlinks)的数目和质量决定。对于数目，一个网页有越多的链接指向它(反向链接数多)，那么表示其他网页对其的认可度就越高，同时这个网页被访问的机会就越大，这样推测出网页的重要性也就越高。对于质量，这个网页如果被很多重要性高的网页所指向，那么其重要性也就越高。如果不考虑质量，就会出现局部最优，而不是全局最优的问题。最典型的就是作弊网页，人为地在一些网页中设置了大量反向链接指向其自身的网页，以提高该网页的重要性。如果不考虑链接质量，就会被这些作弊者所利用。

2. 链接重要度

链接重要度IL(P)是一个关于URL字符串的函数，考察的是字符串本身。链接重要度主要通过一些模式来确认，如认为包含".com"或者"home"的URL重要度高，以及具有较少斜杠的URL重要度高等。

3. 平均链接深度

平均链接深度为ID(P)，表示在一个种子站点集合中，每个种子站点如果存在一条链路(广度优先遍历规则)到达该网页，那么平均链接深度就是一个重要性指标。因为距离种子站点越近，说明被访问的机会越多，因此重要性越高。可以认为种子站点是那些重要性最高的网页，离种子站点越远，重要性越低。事实上，按照广度优先的遍历规则即可满足这种重要性高的网页被优先抓取的需要。

4. 网页重要性度量

网页重要性的度量为$I(P)$，它由以上两个量化值线性决定，即

$$I(P)=\alpha\times \mathrm{IB}(p)\times\beta\times \mathrm{IL}(p)$$

其中α和β是介于0到1之间的参数。

平均链接深度由广度优先的遍历规则保证，因此不作为重要性评价的指标。在抓取能力有限的情况下，如果能够把重要性高的网页尽可能地抓完，是合理科学的，最终被用户查询到的网页也往往是那些重要性高的网页。

除此之外，还要考虑到万维网随时间而动态变化的一面。例如，如何抓取那些新增的网页，如何重访那些被修改了的网页，如何发现那些被删除了的网页。因此，为了保持和万维网网页的同步变化，就必须有网页重访策略。通过该策略可以识别增加、修改及删除网页变化的情况。

3.2.2 深度优先策略

深度优先策略类似于中华民族家族继承的策略，典型的如帝位的继承。通常为长子继承，如果长子过世，长孙的优先级大于次子的优先级。如长孙过世且长孙无子，那么次子继承，这种在继承上的优先关系也称为深度优先策略。

为了方便描述深度优先策略，下面给出一个如图3-2所示的网页链接模型。在这里，假设没有一对网页是互相指向对方的。

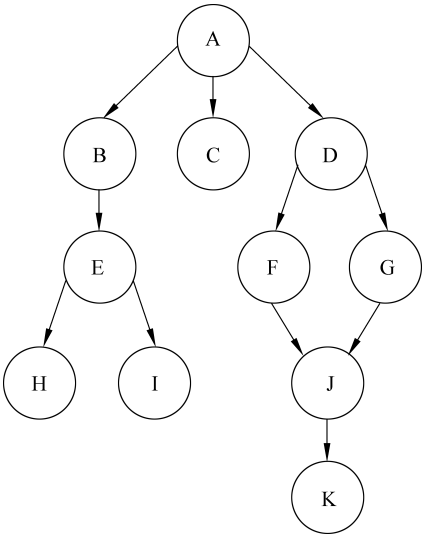

图 3-2　简化的网页链接模型

假设搜索引擎爬虫从 A 出发，根据深度优先的策略，所走的路径为：

① A→B→E→H

② A→B→E→I

③ A→C

④ A→D→F→J→K

⑤ A→D→G→J→K

通过所走的路径，我们看到深度优先的策略是尽量往最远的地方走，直到不能再走为止。但是，我们也发现，爬虫爬行了很多重复的节点，所以一定要有一个较好的算法来控制爬虫爬行的路径避免重复。

在实际应用中，如何确定爬行到停止的程度？一般来说，可以手动来制定爬行的深度。例如，可以制定爬行到 3 层或者 4 层，具体要根据实际情况来决定。

在爬行过程中，爬虫要做一些计算，首先判断“是否要继续向深层爬行”，每次爬行一个链接都要访问一下总链接库，并询问“这个链接是否已经爬行过了”，然后还要记录每次爬行的分支节点，为下次爬行做准备。

另外，如果搜索引擎只是检索简体中文网页，则一旦遇到一个网页的 IP 属于国外 IP，就立即停止，不再进行爬行，否则就有可能爬到国外网站上去了。

3.2.3　广度优先策略

广度优先也称为“层次优先”，它是一种层次型距离不断增大的遍历方法。在图 3-3 中，将这个链接模型划分为 5 层。

第 1 层：A

第 2 层：B、C、D

第 3 层：E、F、G

第 4 层：H、I、J

第 5 层：K

第 1 层优先级最高，第 2 层优先级大于第 3 层，每层的内部优先级从左到右排列。广度优先策略不需要记录上次爬行的分支节点，这一点大大降低了控制的难度。

在抓取策略上，选择广度优先有 3 点优越性。

(1) 重要的网页往往离种子站点距离较近，这符合直觉。通常打开某主要网站的时候，主要内容在主页上，随着不断浏览(可以理解为深度不断加深)，所看到网页的重要性越来越低。

(2) 互联网的深度没有想象的深，到达某一个网页的路径通常很多，总会存在一条很短的路径。

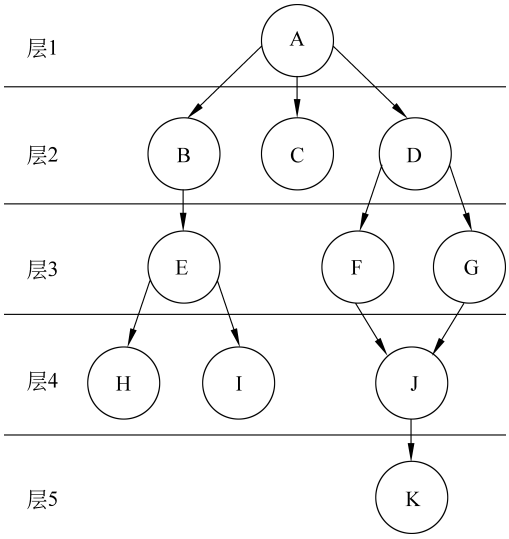

图 3-3　广度优先爬行策略

(3) 广度优先也更适合爬虫的分布式处理,启动多个爬虫,每个爬虫负责一层。

进行广度优先遍历时,必须有一个队列数据结构支持。这个队列可以理解为工作负载队列,只要没有完成抓取任务,就需要提取头部位置的网页继续抓取。直到完成全部抓取任务、工作负载队列为空为止。具体构成如下。

① 爬虫提取工作负载队列中的网页 A(见图 3-3),抓取后,通过对网页 A 的链接分析提取指向网页 B、C、D 的链接放到工作队列中去。此时,工作负载队列中增加了网页 B、网页 C 和网页 D。

② 工作负载队列中的网页 A 抓取完毕,继续抓取工作队列中第 1 个任务,即网页 B。并提取指向网页 E 的链接放到工作队列中。此时工作队列中的排列顺序为:网页 B、C、D、E。

③ 根据广度优先策略,当网页 B 抓取完毕后,爬虫继续抓取网页 C 以及网页 D。当抓取网页 D 时,提取指向网页 F 和网页 G 的链接并放入工作队列中。此时工作队列中的排列顺序为:网页 D、E、F、G。

④ 继续抓取网页 D 的内容,直到结束。然后进行余下的步骤,直到全部抓取完毕。

3.2.4　最佳优先策略

最佳优先策略的算法是按照一定的网页分析算法,预测候选的 URL 与目标网页的相似度,或者是与这个网页的主题的相关性,并选取评价最好的一个或几个 URL 进行抓取。它只访问经过网页分析算法预测“有用”的网页。这种算法是在抓取网页成功以后,要对网页进行相对准确的抽取,然后对次级 URL 和 URL 的相关性进行分析,最后根据这些信息来分析结果,判定哪些链接是下一步抓取的目标。

目前流行的有两大类最佳优先策略:基于立即回报价值评价的搜索抓取策略模式和基于未来回报价值评价的搜索抓取策略模式。前者计算链接的价值的依据主要是在抓取

的过程中“在线”获得的信息，如已访问的页面中的文本信息、链接周围的文本信息和页面之间的结构信息；后者计算链接的价值的主要依据是经预先训练获得的某些经验，用于对远期回报的预测。

基于立即回报价值评价的搜索抓取策略模式的主要算法有鱼群算法(Fish-Search)和鲨鱼算法(Shark-Search)。

鱼群算法是将用户输入的描述信息切分成关键字和短语后作为主题，将其下一级抓取链接的描述信息看作是与主题相关的。缺点是不能评价相关程度的高低，有局限性。例如，把 Web 的爬取看作是鱼群觅食的过程，鱼就相当于网站的 URL，食物则相当于相关的网页。我们可以动态地建立一个优先爬取的 URL 列表，相关度用 0、1 表示，0 表示不相关，1 表示相关，对此只能做与主题相关与不相关的判断，而不能获得相关程度高低的评价。以下为鱼群算法计算相关度的公式。

$$\mathrm{sim}(q,p)=\frac{\sum w_{qk}\times w_{pk}}{\sqrt{\sum w_{qk}^2\sum w_{pk}^2}}$$

其中，q 表示主题切分后的关键字集合，代表页面链接文本集合；w_{tk} 表示 t 中单词 k 对某一主题的重要程度，t 可以是 q，也可以是 p。通常 w_{tk} 由下面的公式计算。

$$w_{tk}=\mathrm{TF}_{tk}\times\mathrm{IDF}_{tk}=\frac{\mathrm{tf}(tk,p)}{\sum_{i=1}^{n}\mathrm{tf}(ti,p)}\times\log\left(\frac{N}{nk}\right)$$

其中，$\mathrm{tf}(tk,p)$是主题 tk 在 p 中出现的频率，N 为文档集中的所有文档数，nk 为 tk 中出现的文档数。这样就可以判断是否与主题相关。

鲨鱼算法是一种基于鱼群算法的改进，优点是可以获得一个 URL 与主题的相关程度。

(1) 判断与父节点的相关性的公式。

$$\mathrm{Inherited}(\mathrm{child}_{\mathrm{url}})=\begin{cases}\delta\times\mathrm{sim}(q,\mathrm{current}_{\mathrm{url}}), & \mathrm{sim}(q,\mathrm{current}_{\mathrm{url}})<0\\ \delta\times\mathrm{inherited}(\mathrm{current}_{\mathrm{url}}), & \text{其他}\end{cases}$$

其中，$\delta<1$，表示一个常量。

(2) 通过链接文本计算链接上下文相关程度的公式。

$$\mathrm{anchor}_{\mathrm{context(url)}}=\begin{cases}1, & \mathrm{anchor(url)}>0\\ \mathrm{sim}(q,\mathrm{anchor}_{\mathrm{text}}), & \text{其他}\end{cases}$$

当 $\mathrm{anchor(url)}>0$ 时它的值都是等于 1 的，其他时候它的值等于 $\mathrm{sim(q,anchor}_{\mathrm{text}})$，这样通过对一个鱼群算法的改进，就可以获得 URL 与主题的相关程度。

3.2.5 不重复抓取策略

不重复的关键在于爬虫记住爬行的历史，只有记住过去才可能不重复。爬虫记录历史的方式是哈希表(也称为“杂凑表”)，每一条记录是否被抓取的信息存放在哈希表的某一个槽位上。如果某网页在过去的某个时刻已经被抓取，则将其对应的槽位的值置 1；反之置 0。而具体映射到哪一个槽位，则由哈希函数决定。

1. MD5 签名函数

在介绍哈希表前，首先简单了解一下 MD5 签名函数。MD5 签名是一个哈希函数，可以将任意长度的数据流转换为一个固定长度的数字（通常为 4 个整型数，即 128 位）。这个数字称为“数据流的签名”或者“指纹”，并且数据流中的任意一个微小的变化都会导致签名值发生变化。

将 URL 字符串数字化是通过某种计算将任何一个 URL 字符串唯一地计算成一个整数。在一个 URL 哈希函数映射下，任意一个字符串都唯一地对应一个整数。一个 64 位整数可以表达 1800 万 TB(1TB=1024GB)，而字符串空间的大小远远大于 64 位整数所表达的整数空间大小，因此碰撞是不可避免的。碰撞是指那些字符串不同，而计算出相同的签名值的情况。然而如果哈希函数设计得足够好，则相互碰撞的机会可以小到忽略不计。

标准 MD5 签名的整数空间很大，128 位整数能表达 2^{128} 个不同的数，这是十分巨大的，而实际分配的哈希表空间十分有限，一个普通 32 位处理器，理论上最多可以分配 2^{32} 大小的内存，即 4GB 大小的内存。

因此，在实际处理中将签名值进行模运算映射到实际的哈希表中（可以理解为哈希表存放在内存中）。因此实际的哈希函数是 MD5(URL)%n（%表示取模运算），这样使得一个 URL 被映射到大小为 n 的哈希表的某个槽位上。

2. 哈希表

哈希表是简单的顺序表，即数组。从实际应用的角度看，这个数组要足够大，而且尽可能地全部放在内存中，以保证每一个 URL 的签名都可以通过查找哈希表来确定是否曾经抓取过，如图 3-4 所示。

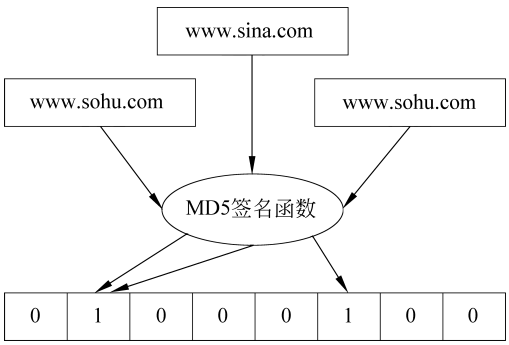

图 3-4　哈希函数示例

假定抓取顺序为 www.sohu.com、www.sina.com 和 www.sohu.com。首先在抓取 www.sohu.com 时为该 URL 做签名运算，然后取模，假定得到的结果为 2，存放在数组的第 2 个槽上。这个过程只要将第 2 个槽位置 1，表示已访问过。继续抓取 www.sina.com，以同样的方法，假定将其存放在数组的第 6 个槽位上，之后重复抓取到了 www.sohu.com。由于 MD5 签名函数的特性，任何一个自变量唯一对应一个因变量，因此计算 www.sohu.com 的签名，同样得到结果为 2。当查询第 2 个槽位时，发现该槽位已经置 1，说明已经访问过；因此不再重复访问。

目前中文网页总数已经超过2000亿，如果要存放2000多亿网页是否被抓取过这样一个历史信息，那么需要的哈希表将会非常大。如果哈希表每一个单元为1B(8位)，则至少需要10GB的容量。而且为了保证碰撞的概率极低，实际需要的容量远大于10GB。对于32位操作系统来说，最大寻址空间为4GB，即理论上最大的物理内存为4GB，因此在内存中不可能分配一个如此巨大的哈希表。

为了能够将整个哈希表装入内存，通常采用Bitmap的数据结构压缩内存用量，Bitmap需要借助按位与(&)和按位或(|)，以及左移(≪)和右移(≫)这4种基本位运算。

3. Bitmap结构

Bitmap结构使用整型作为基本单元；一个整型为32位；一个int Hash[8]数组仅通过8个单元来记录历史。而如果将比特作为最小单元，则能扩展到256个单元。

例如，定义这样一个哈希表，即int Hash[8]。将“www.sohu.com”作为字符串执行一次MD5签名，假定得到的签名值为34。

用34除以32(一个整型为32位)，得到商数为1，然后用1对8取模运算得到1。此结果表示槽位在Hash[1]中。

用34除以32，得到余数为2，表示34映射到Hash[1]这个32位整数的第3个比特位上(从低位算起，余数为0映射到第1个比特位，余数为1映射到第2个比特位，以此类推)。

将第3个比特位置1，则Hash[1]的值为4，这是因为Hash[1]的第3个比特位被置1。

将34按比特位的方式展开如图3-5所示。数值34的二进制的前3位001决定了它存在数组的第2个位置上(数组的起始下标为0)，也就是数组Hash[1]中。前面提到的用34除以32后，对8取模，目的就是取出数值34的前3位。后5位为00010，表示存放在数组Hash[1]的第3个比特位上。前面提到的34除以32后得到的余数就是这里的后5位。

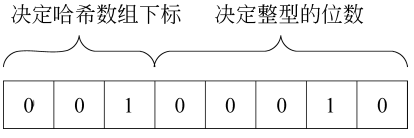

图3-5 数值34的比特位表示

通过前面的计算，Hash[1]的实际值为4，表示其第3个比特位被置1。这样在Hash[1]中用4这个数值完整地表示了34这个签名值。

通过Bitmap结构，利用计算机快速地按位与、按位或和移位运算能够高效地实现记住历史、不重复抓取的策略，同时也将原来的哈希表压缩了1/32。

4. Bloom filter算法

一个网址是否被访问过，最直接的方法就是将集合中全部的元素存在计算机中，遇到一个新元素时，将它和集合中的元素直接比较即可。一般来讲，计算机中的集合是用哈希表来存储的。它的好处是快速准确，缺点是费存储空间。当集合比较小时，这个问题不显

著，但是当集合巨大时，哈希表存储效率低的问题就显现出来了。例如，全世界少说也有几十亿个网站的地址，将它们都存起来则需要大量的网络服务器。如果用哈希表，每存储一亿个网站地址，就需要 1.6GB 的内存。用哈希表实现的具体办法是将每一个网站地址对应成一个 8B 的信息指纹，然后将这些信息指纹存入哈希表。由于哈希表的存储效率一般只有 50%，因此一个网站地址需要占用 16B。一亿个地址大约要 1.6GB，即 16 亿字节的内存。因此存储几十亿个网站地址可能需要上百 GB 的内存。除非是超级计算机，一般服务器是无法存储的。

Bloom filter(布隆过滤器)是由巴顿·布隆于 1970 年提出的。它实际上是一个很长的二进制向量和一系列随机映射函数。下面通过一个例子来说明其工作原理。

假定存储一亿个网站地址，先建立一个 16 亿二进制(比特)，即 2 亿字节的向量，然后将这 16 亿个二进制全部设置为零。对于每一个网站地址 X，用 8 个不同的随机数产生器(F_1，F_2，…，F_8) 产生 8 个信息指纹(f_1，f_2，…，f_8)。再用一个随机数产生器 G 把这 8 个信息指纹映射到 1～16 亿中的 8 个自然数(g_1，g_2，…，g_8)。现在把这 8 个位置的二进制全部设置为 1。当对这 1 亿个网站地址都进行这样的处理后，一个针对这些网站地址的 Bloom filter 就建成了，如图 3-6 所示。

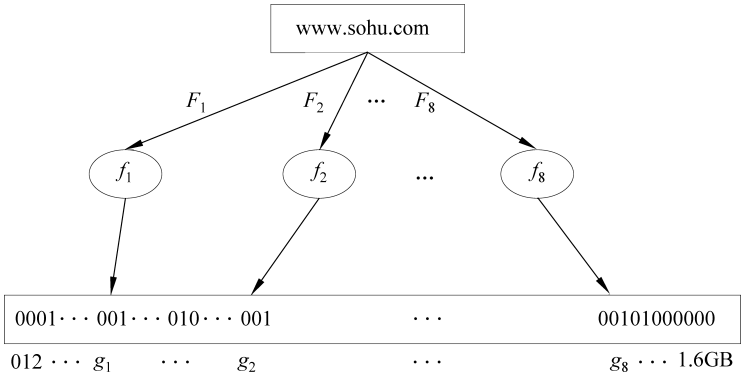

图 3-6 Bloom filter 原理

用相同的 8 个随机数产生器(F_1，F_2，…，F_8)对这个地址产生 8 个信息指纹 (s_1，s_2，…，s_8)，然后将这 8 个指纹对应到 Bloom filter 的 8 个二进制位，分别是 t_1，t_2，…，t_8。如果网站 Y 在其中，显然，t_1，t_2，…，t_8 对应的 8 个二进制一定是 1。这样就能准确地发现网站 Y 了。

Bloom filter 不会漏掉任何一个爬行过的地址。但是，它有可能将一个不在名单中的网站地址判定为在名单中，因为有可能某个网站的地址正巧对应 8 个都被设置成 1 的二进制位。好在这种可能性很小，我们把它称为误识概率。一般情况下，误识概率在万分之一以下。

利用 Bloom filter 算法可以更加经济地利用好哈希表中的比特位，使得更加经济地、灵活地使用内存。下面通过图 3-7 来进一步理解。对于一个字符串独立地使用 m 哈希函数计算，然后将它映射到哈希表的 m 槽上。如果这 m 个槽都置 1，说明该字符串已经在历史上出现过；否则说明该字符串是第 1 次出现，将 m 个槽都置 1。

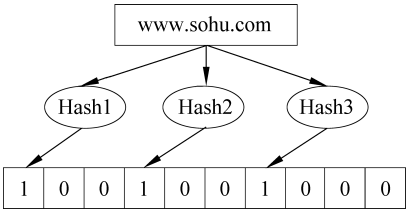

图 3-7 Bloom filter 算法原理

在抓取到 www.sohu.com 时，假设执行 3 次独立的不同的哈希函数（Hash1、Hash2 和 Hash3）运算，映射到哈希表的 3 个槽上。假定是 1、4 和 7 槽位。接下来，如果爬虫又抓取到了 www.sina.com，经过同样的 3 次哈希函数运算，假定计算的结果是 1、4 和 6，很明显可以看出槽 6 是之前没有被置位的，也就是说，www.sina.com 没有被抓取过。成功抓取 www.sina.com 后，将槽位 6 置 1。Bloom filter 算法一方面提高了哈希数组的利用效率，并被证明适宜于应用在大规模集合查找计算中，同时在网络应用中也被大量采用。

5. 不重复抓取策略的使用

在抓取过程中，可以使用多个爬虫来合作抓取，这样可以进一步降低每个爬虫的用于记录历史抓取情况的哈希表大小。如果有 n 个爬虫，则可将哈希表继续压缩到原有大小的 n 分之一。如果 n 个爬虫分别运行在不同的机器上，那么每个机器被哈希表占用的内存用量将非常少。通常保持抓取历史记录所需要的内存在百兆字节左右是恰当的。

通过不重复抓取的方法初步解决了死循环的问题，即抓过的不再抓。然而实际操作中还有这样一个问题，如果任意两个网页存在链接，则链接它们的最短路径值为 17。这样，爬虫无论用何种遍历方法都不能保证一定会按照最佳路径抓取每一个网页，因为任何一个网页都可能从多个种子站点开始广度优先被遍历到。

为了防止爬虫无限制地广度优先抓取，必须在某个深度上进行限制。到达这个深度后就应该停止抓取，这个深度的取值就是万维网直径长度。当在最大深度上停止时，那些深度过大的未抓网页，总是期望可以从其他种子站点更加经济地到达。例如，种子站点 B 和 C 在抓取到深度为 17 的时候，立即停止抓取，把抓取剩余网页的机会留给从种子站点 A 出发的进行抓取工作的爬虫。此外，深度优先策略和广度优先策略的组合可以有效地保证抓取过程中的封闭性。即在抓取过程（遍历路径）中总是在抓取相同域名下的网页，而很少出现其他域名下的网页。

3.2.6 网页重访策略

网络爬虫的任务就是不断地下载各种各样的网页，但是，网页是在不断变化的，刚刚下载的网页有可能变化了，因此爬虫不得不周期性地刷新，重访那些已经下载的网页。通过重访以使得这些网页能够与万维网的变化一致。目前，网页重访策略大致可以归为以下两类。

1. 统一的重访策略

网络爬虫用同样的频率重访那些已经抓取的全部网页，以获得统一的更新机会，所有的网页不加区别地按照同样的频率被爬虫重访。

2. 个体的重访策略

不同网页的改变频率不同，爬虫根据其更新频率来决定重访该个体页面的频率。即对每一个页面都量身定做一个爬虫重访频率，并且网页的变化频率与重访频率的比率对

任何个体网页来说都是相等的。即对于任意网页 P_i，其固有的更新频率为 λ_i（次数/单位时间），爬虫对其重访的频率为 f_i，则 λ_i/f_i 为一个常数。即对于网页 P_i 和 P_j，有 $\lambda_i/f_i=\lambda_j/f_j(i\neq j)$。那些更新频率高的网页，重访频率就高，更新频率低的网页，重访频率就低。

研究表明，网页的更新过程符合泊松过程，网页更新时间间隔符合指数分布，因此，可以对不同类型的网页采用不同的更新策略。

泊松分布是一种统计与概率学里常见到的离散概率分布，由法国数学家西莫恩·德尼·泊松（Siméon-Denis Poisson）在1838年时发表。泊松分布的概率密度函数为：

$$P(X=k)=e^{-\lambda}\frac{\lambda^k}{k!}\quad k=0,1,2,\cdots$$

其中，$\lambda>0$ 是常数，则称 X 服从参数为 λ 的泊松分布。泊松分布的参数 λ 是单位时间（或单位面积）内随机事件的平均发生率。泊松分布适合于描述单位时间内随机事件发生的次数。例如，某一服务设施在一定时间内到达的人数、电话交换机接到呼叫的次数、汽车站台的候客人数、机器出现的故障数、自然灾害发生的次数，等等。

3.2.7 网页抓取提速策略

1. 策略

网页抓取提速策略又称为合作抓取策略，通常可以采用下面几种方法。

（1）提高抓取单个网页的速度。

（2）尽可能减少不必要的抓取任务。

（3）增加同时工作的爬虫数量。

对于第（1）种方法，单个网页的抓取速度受到下载带宽的限制，在现有技术条件下很难提高。第（2）种方法难度比较大，由于需要和万维网的变化保持紧密同步，如果要减少不必要的抓取，将会导致网页重访不及时，这样就不能快速同步目标网页的变化。第（3）种方法通过增加爬虫数量，是目前广泛使用的方法。

2. 任务分解

在多个爬虫抓取的情况下，就要解决一个网页交给哪一个爬虫抓取的问题。如果分工不明，很可能多个爬虫抓取了相同的网页，从而引入额外的开销。通常采用以下两种方法来进行抓取任务的分解。

（1）通过 Web 主机的 IP 地址来分解，使某个爬虫仅抓取某个地址的网页。

（2）通过网页的域名来分解，使某个爬虫仅抓取某个域名段的网页。

对于大型网站来说，通常采用负载均衡的 IP 组技术，即一个域名对应于多个 IP 地址技术；对于小型网站来说，通常采用多个域名对应一个 IP 地址的技术。鉴于多域名对应相同 IP 和同域名对应多个 IP 的情况，通常的做法是按照域名分解任务，即只要保证不重复抓取大型网站的网页、小型网站即便重复抓取也可以接受的策略分配任务。这种分配方法将不同的域名分配给不同的爬虫抓取，某一个爬虫只抓取“指定”的一个域名集合下的网页。

3. 调度

为了使按照域名分解的策略更加合理，在下载系统中，按照域名分解抓取任务（网页

集合)的工作由一个称为“调度员”的模块来处理,通过域名分解将不同的网页调度给不同的爬虫进行抓取。因此,下载系统主要由爬虫和调度员构成。

形式化的调度分配方式如下。

首先假定有 n 个爬虫可以并行工作,并且定义一个可以提取 URL 域名的函数 domain。具体过程如下。

(1) 对于任意的 URL,利用 domain 函数提取 URL 的域名。

(2) 用 MD5 签名函数签名域名:MD5(domain(URL))。

(3) 将 MD5 签名值对 n 取模运算:int spider_no=MD5(domain(URL))%n。

(4) 该 URL 分配给编号为 spider_no 的爬虫进行抓取。

3.2.8 Robots 协议

Robots 协议是 Web 站点和搜索引擎爬虫交互的一种方式,robots.txt 是存放在站点根目录下的一个纯文本文件。该文件可以指定搜索引擎爬虫只抓取指定的内容,或者是禁止搜索引擎爬虫抓取网站的部分或全部内容。当一个搜索引擎爬虫访问一个站点时,它会首先检查该站点根目录下是否存在 robots.txt,如果存在,搜索引擎爬虫就会按照该文件中的内容来确定访问的范围,如果该文件不存在,那么搜索引擎爬虫就沿着链接抓取。

另外,robots.txt 必须放置在一个站点的根目录下,而且文件名必须全部小写。

如果搜索引擎爬虫要访问的网站地址是 http://www.w3.org/,那么 robots.txt 文件必须能够通过 http://www.w3.org/robots.txt 打开并看到里面的内容。

```
#robots.txt for http://www.w3.org/
#
#$Id: robots.txt,v 1.84 2019/03/22 22:48:24 gerald Exp $
#

#For use by search.w3.org
User-agent: W3C-gsa
Disallow: /Out-Of-Date

User-agent: W3T_SE
Disallow: /Out-Of-Date

User-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT; MS Search 4.0 Robot)
Disallow: /

#W3C Link checker
User-agent: W3C-checklink
Disallow:

#various other access-controlled or expensive areas
```

```
Disallow: /2004/ontaria/basic
Disallow: /Team/
Disallow: /Project
Disallow: /Web
Disallow: /Systems
Disallow: /Out-Of-Date
Disallow: /2005/06/blog/
Disallow: /2004/08/W3CTalks
Disallow: /2007/11/Talks/search
Disallow: /People/all/
Disallow: /RDF/Validator/ARPServlet
Disallow: /RDF/Validator/rdfval
Disallow: /2003/03/Translations/byLanguage
Disallow: /2003/03/Translations/byTechnology
Disallow: /2005/11/Translations/Query
Disallow: /2000/06/webdata/xslt
Disallow: /2000/09/webdata/xslt
Disallow: /2005/08/online_xslt/xslt
Disallow: /Bugs/
```

具体使用格式如下。

(1) User-agent：用于描述搜索引擎爬虫的名字，在 robots.txt 文件中，如果有多条 User-agent 记录，说明有多个搜索引擎爬虫会受到该协议的限制，对该文件来说，至少要有一条 User-agent 记录。如果该项的值设为 *，则该协议对任何搜索引擎爬虫均有效，在 robots.txt 文件中，User-agent：* 这样的记录只能有一条。

(2) Disallow：用于描述不希望被访问到的一个 URL，这个 URL 可以是一条完整的路径，也可以是部分的，任何以 Disallow 开头的 URL 均不会被 Robot 访问到。

下面举例来说明 robots.txt 的具体用法。

例 1 通过/robots.txt 禁止所有搜索引擎爬虫抓取/bin/cgi/目录，以及/tmp/目录和/foo.html 文件，设置方法如下。

```
User-agent: *
Disallow:/bin/cgi/
Disallow:/tmp/
Disallow:/foo.html
```

例 2 通过/robots.txt 只允许某个搜索引擎抓取，而禁止其他的搜索引擎抓取。例如，只允许名为 slurp 的搜索引擎爬虫抓取，而拒绝其他的搜索引擎爬虫抓取/cgi/目录下的内容，设置方法如下。

```
User-agent: *
Disallow:/cgi/
User-agent:slurp
Disallow:
```

例 3 禁止任何搜索引擎抓取自己的网站，设置方法如下。

```
User-agent: *
Disallow:/
```

例 4 只禁止某个搜索引擎抓取自己的网站。例如，只禁止名为 slurp 的搜索引擎爬虫抓取，设置方法如下。

```
User-agent:slurp
Disallow:/
```

搜索引擎爬虫必须要遵守 Robots 协议并执行 Web 站点的要求，因此搜索引擎爬虫需要有一个分析 Robots 协议的模块，并严格按照 Robots 协议的规定抓取 Web 主机允许访问的目录和网页。

当然，robots.txt 只是一个协议，如果搜索引擎爬虫的设计者不遵循这个协议，网站管理员也无法阻止搜索引擎爬虫对于某些页面的访问，但一般的搜索引擎爬虫都会遵循这些协议，而且网站管理员还可以通过其他方式来拒绝网络爬虫对某些网页的抓取。

搜索引擎爬虫在下载网页的时候，会去识别网页的 HTML 代码，在其代码的部分会有 META 标识。通过这些标识，可以告诉搜索引擎爬虫本网页是否需要被抓取，还可以告诉搜索引擎爬虫本网页中的链接是否需要被继续跟踪。例如，表示本网页不需要被抓取，但是网页内的链接需要被跟踪。

现在一般的网站都希望搜索引擎能更全面地抓取自己网站的网页，因为这样可以让更多的访问者能通过搜索引擎找到此网站。为了让本网站的网页更全面地被抓取到，网站管理员可以建立一个网站地图，即 Site Map。许多搜索引擎爬虫会把 sitemap.htm 文件作为一个网站网页爬取的入口，网站管理员可以把网站内部所有网页的链接放在这个文件里面，那么搜索引擎爬虫就可以很方便地把整个网站抓取下来，避免遗漏某些网页，也会减小网站服务器的负担。

小　　结

1. 搜索引擎爬虫

网络爬虫，也称为蜘蛛程序(Spider)，是一个自动提取网页的程序，是搜索引擎的重要组成部分。爬虫的作用是为搜索引擎抓取大量的数据，抓取的对象是整个互联网上的网页。爬虫程序不可能抓取所有的网页，因为在抓取的同时，Web 的规模也在增大，所以一个好的爬虫程序一般能够在短时间内抓取更多的网页。

网络爬虫在搜索引擎中占有重要位置，对搜索引擎的查全、查准都有影响，决定了搜索引擎数据容量的大小，而且网络爬虫的好坏直接影响搜索结果页中的死链接的个数。

在爬虫开始的时候，需要给爬虫输送一个 URL 列表，这个列表中的 URL 地址便是爬虫的起始位置，爬虫从这些 URL 出发，开始爬行，一直不断地发现新的 URL，然后再根据策略爬行这些新发现的 URL，如此永远反复下去。

对于网络爬虫来说，抓取下来的网页包括各种格式，如 HTML、图片、DOC、PDF、多

媒体、动态网页及其他格式等。这些文件抓取下来后，需要把这些文件中的文本信息提取出来。准确提取这些文档的信息，一方面对搜索引擎的搜索准确性有重要作用，另一方面对于网络爬虫正确跟踪其他链接有一定影响。

搜索引擎的更新周期对搜索引擎搜索的查全率有很大影响。如果更新周期太长，则总会有一部分新生成的网页搜索不到；周期过短，技术实现会有一定难度，而且会对带宽、服务器的资源都有浪费。

2. 爬虫使用的关键技术

网页重要性度量由链接欢迎度、链接重要度和平均链接深度这3个方面决定。

网络爬虫采取的抓取策略主要有：深度优先策略、广度优先策略、不重复抓取策略、网页抓取优先策略、最佳优先策略、网页重访策略和网页抓取提速策略。

Robots协议是Web站点和搜索引擎爬虫交互的一种方式，robots.txt是存放在站点根目录下的一个纯文本文件。该文件可以指定搜索引擎爬虫只抓取指定的内容，或者是禁止搜索引擎爬虫抓取网站的部分或全部内容。

思　考　题

1. 搜索引擎中的爬虫是如何工作的？

2. 除了书中介绍的开源爬虫之外，还有哪些搜索引擎的爬虫？它们有什么特点？

3. 从网上下载JSpider爬虫，抓取指定的一个网站，然后分析下载后网页的特点。

4. 简述深度优先策略和广度优先策略，并比较它们的特点。

5. 除了普通搜索引擎之外，还有很多特殊的搜索引擎。上网查找资料，简述网页库级垂直搜索引擎所使用的技术及其特点。

6. 简述布隆过滤器的基本工作原理。

7. 编写robots.txt文件，禁止所有搜索引擎爬虫抓取/main/目录，以及/www/目录下的index.html文件。