第141封信 | 如何下载整个互联网上的全部网页？

吴军

第141封信 | 如何下载整个互..
11:28 5.37MB

信件朗读者：宝木

小师弟，你好！

有了昨天的铺垫，我们今天讲讲如何构建一个网络爬虫，下载整个互联网上全部的网页。

对于这道问题，有些面试者会简单地回答道："这很简单。整个互联网是彼此连通的，从任何一个网页出发，跟随它的超链接采用深度优先算法，或者广度优先算法，就可以下载整个互联网了。"

这个道理是对的，事实上你如果用开源的软件，从一家门户网站比如雅虎的首页出发，先下载这个网页，然后通过分析这个网页，可以找到页面里的所有超链接，也就等于知道了这家门户网站首页所直接链接的全部网页，诸如雅虎邮件、雅虎财经、雅虎新闻等。接下来访问、下载并分析这家门户网站的邮件等网页，又能找到其他相连的网页。让计算机不停地做下去，就能下载很多很多网页。

但是，如果一个候选人对图论或者网络爬虫的理解仅仅到这一步，他一定无法搭建起一个实用的程序，当然也无法通过Google的面试。因为他忽略了太多的实际问题。对于这道面试题，我一般会按照下面的次序一步步追问。

首先我们要知道，现在的互联网非常庞大，今天Google的索引中有超过1万亿个网页，即使更新最频繁的基础索引也有几百亿个网页，假如下载一个网页需要一秒钟，下载这100亿个网页则需要317年。如果下载10000亿个网页则需要32000年左右，是我们人类有文字记载历史的5到6倍的时间，这显然不现实。

顺便说一句，下载每一个网页因为需要有网络通信的时间，网速再快这个时间也省不了，因此时间不会太短，即便缩短10倍，100亿个网页花31.7年或者1万亿个网页花3200年也是一个不可接受的时长。

当然，有点计算机工作经验的人会马上想到使用并行计算，比如让1000台服务器同时工作，这样下载100亿个网页大约只需要半个月的时间，这是可以接受的，当然，下载1万亿个网页还有点挑战性。然而，世界上任何事情都是有成本的，并行计算也是如此。

为了保障这1000台服务器不会把某个网页访问好几次，而把另外一些网页漏掉，需要有一个小本本，记录已经下载过的网页。这个小本本需要多大呢？如果每一个网页写上一笔只需要8个字节，那么存100亿个网页的记录也需要80GB的内存，今天还没有哪台通用的服务器有这么大的内存。即使有，如果我们把这个关于已经下载网页记录的"小本本"放在一台服务器上，所有其它服务器在决定是否下载某个网页之前先要询问它一次，它也一定会成为瓶颈。然而，如果不把"小本本"集中管理，而是分散到各个服务器中，那么每一次下载，都要询问所有的服务器，也显然不现实。

因此，这道看起来很简单的面试题，里面有很多玄机。事实上，一个商业的网络爬虫需要有成千上万个服务器，并且通过高速网络连接起来。如何建立起这样复杂的网络系统，如何协调这些服务器的任务，就是网络设计和程序设计的艺术了。

至于小本本怎么管理，其实就要用到我在写给你的第64封信中，讲到的随机化的原理了，**随机化之后看上去无序，其实反而变得有规律可循，于是每个服务器看到一个网页，都知道这个网页是否属于自己的工作范围，如果不属于，则通知相应的服务器去处理，而不需要到处广播。**

再接下来，我常常会问候选人，图论中给出了广度优先遍历（BFS）和深度优先遍历（DFS）两种算法，到底该用哪个呢？

虽然从理论上讲，这两个算法（在不考虑时间因素的前提下）都能够在大致相同的时间里"爬下"整个"静态"互联网上的内容，但这只是理论上的可行性，它有两个假设——不考虑时间因素，互联网静态不变，都是现实中做不到的。

搜索引擎的网络爬虫问题更应该定义成"**如何在有限时间里最多地爬下最重要的网页**"。显然各个网站最重要的网页应该是它的首页。在最极端的情况下，如果爬虫非常小，只能下载非常有限的网页，那么应该下载的是所有网站的首页，如果把爬虫再扩大些，应该爬下从首页直接链接的网页（就如同和北京直接相连的城市），因为这些网页是网站设计者自己认为相当重要的网页。在这个前提下，显然BFS明显优于DFS。

事实上在搜索引擎的爬虫里，虽然不是简单地采用BFS，但是先爬哪个网页，后爬哪个网页的调度程序，原理上基本上是BFS。

那么是否DFS就不使用了呢？也不是这样的。这是和爬虫的分布式结构以及网络通信的握手成本有关。**所谓"握手"就是指下载服务器和网站的服务器建立通信的过程**。这个过程需要额外的时间（Overhead Time），如果握手的次数太多，下载的效率就降低了。

实际的网络爬虫都是一个由成百上千甚至成千上万台服务器组成的分布式系统。对于某个网站，一般是由特定的一台或者几台服务器专门下载。这些服务器下载完一个网站，然后再进入下一个网站，而不是每个网站先轮流下载5%，然后再回过头来下载第二批。这样可以避免握手的次数太多。如果是下载完第一个网站再下载第二个，那么这又有点像DFS，虽然下载同一个网站（或者子网站）时，还是需要用BFS的。

总结起来，网络爬虫对网页遍历的次序不是简单的BFS或者DFS，而是有一个相对复杂的下载优先级排序的方法。管理这个优先级排序的子系统一般称为调度系统（Scheduler），由它来决定当一个网页下载完成后，接下来下载哪一个。

当然在调度系统里需要存储那些已经发现但是尚未下载的网页的URL，它们一般存在一个优先级队列（Priority Queue）里。而用这种方式遍历整个互联网，在工程上和BFS更相似。因此，在爬虫中，BFS的成分多一些。

如果一个候选人能把这些因素都考虑进去，他已经通过我的面试了。但是，如果我有时间，还会继续追问一个看似简单，但在工程上很复杂的问题，那就是如何分析一个HTML页面，然后把里面的超链接地址URL提取出来？

这件事在过去并不难，因为过去的网站都是明码编写这些URL，前后都有明显的标识，很容易提取出来。但是现在很多URL的提取就不那么直接了，因为很多网页如今是用一些脚本语言（比如JavaScript）生成的。打开网页的源代码，URL不是直接可见的文本，而是运行这一段脚本后才能得到的结果。

因此，网络爬虫的页面分析就变得复杂很多，它要模拟浏览器运行一个网页，才能得到里面隐含的URL。有些网页的脚本写得非常不规范，以至于解析起来非常困难。可是，这些网页还是可以在浏览器中打开，说明浏览器可以解析。因此，**需要做浏览器内核的工程师来写网络爬虫中的解析程序，**可惜出色的浏览器内核工程师在全世界数量并不多。

除此之外，建立互联网的下载爬虫还有很多技术细节，比如有些网站网页更新快，如何处理？有些几乎是静态的网页，平时没有更新，又该如何处理等等。这些因为篇幅的问题我就不介绍了。

总之，"如何构建一个网络爬虫"是我在Google最常使用的一道面试题，虽然很多候选人知道我要问他们这个问题，但是依然很难回答得圆满，因为这个问题太开放了，我能找到他们显示出的弱点，不断追问下去。

当然，我不断提问不是为了为难哪个候选人，而是为了有效地考察出一个候选人的计算机科学理论基础、算法能力和他的工程素养。而我给候选人的打分也未必和他的感觉一致，对有些人我问得比较深，是因为我已经判断出基本的原理他都懂了，不必要问了，他即使觉得没有回答完美，我的打分可能也很高。但是，如果一个仅仅有执行能力，但对这个问题有太多地方没有考虑到，我可能就不会往深了问，因此即使他们觉得都答上来，也不会通过面试。