Web信息处理与应用

第二节

爬虫基础

徐童

2020.9.24

上节回顾

- 1965-1989，两位先驱的成果

- Ted Nelson在1965年提出了超文本的概念。
  - HyperText，源自于“非连续性著述”（Non sequential writing）的理念，即分叉的、允许读者作出选择的、无限延伸扩展的非线性文本。
- 1989年，Tim Berners Lee（万维网之父）等人首次提出了一个World Wide Web协议，随后定义了URL、HTML、HTTP等的规范，使网络能够为大众所使用。

上节回顾

- 1993年，脱胎于爬虫的搜索引擎Wanderer

- Wanderer：最早的爬虫
  - 由MIT的学生Matthew Gray设计
  - 原意用于统计互联网上服务器的数量，而非为搜索引擎所设计
- Wandex：最早的网页索引计划
  - Wanderer后来发展为可以捕获网址，而为这些网址建立索引的计划就是Wandex

Wandex

本节主题

- 本课程所要解决的问题

第一个问题：

Web信息如何获取?

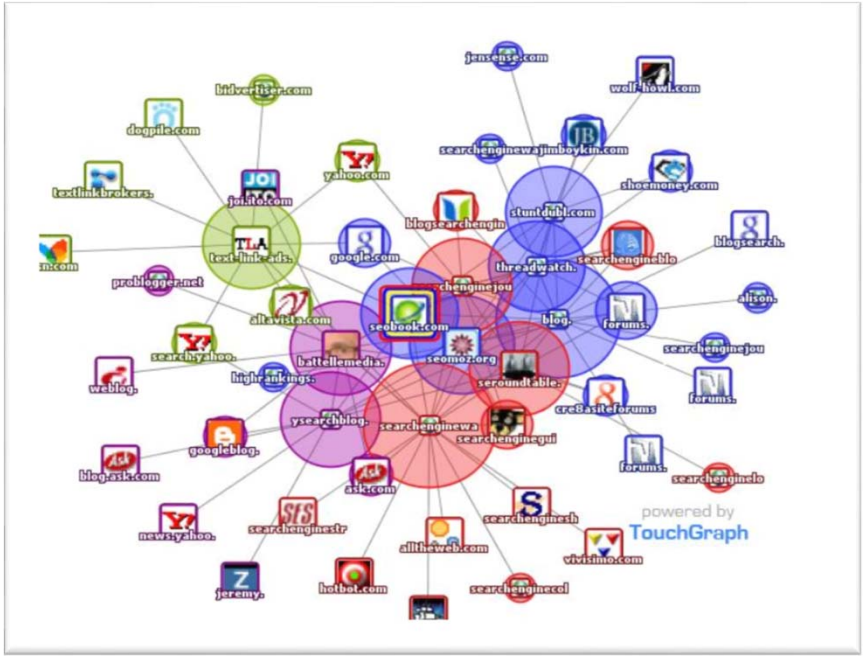
推荐课程

- **本节课程说明**

- 本节课以爬虫基本概念和基础知识为主
  - 计算机课？语文课！（滑稽  ）
- 有想要了解更多关于爬虫实战内容的同学，可参考以下在线课程：
  - Python教程（第21节-第29节部分）
    - https://www.bilibili.com/video/BV1ws411i7Dr/
  - Python爬虫全套课程
    - https://www.bilibili.com/video/BV1Yh411o7Sz

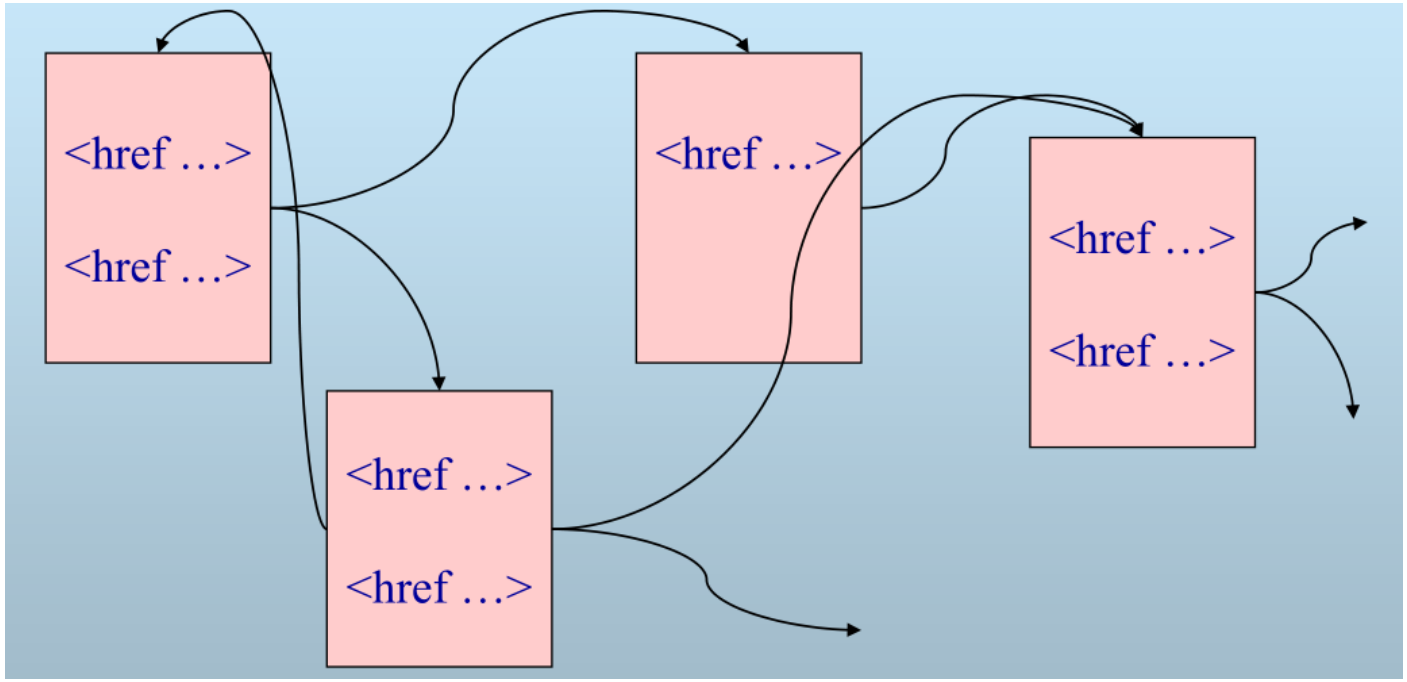
本节目录

- **网络爬虫的定义与需求**
- 爬虫的基本要素
- 面向API的新爬虫任务
- 常见的爬虫算法
- 常见反爬虫机制与应对策略
- 分布式爬虫简述



如果将互联网视为主机构成的网络，万维网则无疑可视为文档的网络

- **Web网络的图模型**
- 以网页为节点、超链接（Hyper-Link）为有向边



- 网页间的互联互通

以科大主页
作为种子节点

中国科学技术大学
University of Science and Technology of China
理论学习网

求是网

CCTV.com 央视网

各政务机构，e.g., 全国人大

各中央媒体，e.g., 人民日报

各地方媒体，e.g., 南方网

各友情链接，e.g., 前程无忧

- **爬虫的任务定义**

- 从一个种子站点集合（Seed sites）开始，从Web中寻找并且下载网页，获取排序需要的相关信息，并且剔除低质量的网页。
- 常见的爬虫类型
  - 通用网络爬虫：目标为全网Web信息，主要为门户站点搜索引擎和大型Web 服务提供商采集数据。
  - 聚焦网络爬虫：选择性爬取与预定主题相关的内容。
  - 增量式网络爬虫：对已下载内容进行增量式更新并只爬取更新内容。

- **爬虫的任务定义**

- 从一个种子站点集合（Seed sites）开始，从Web中寻找并且下载网页，获取排序需要的相关信息，并且剔除低质量的网页。
- 常见的爬虫类型
  - 深度网络爬虫：又称暗网（Deep Web）爬虫，专门负责获取搜索引擎无法索引的、超链接不可达的或需提交表单后才可见的网络内容。

- **爬虫的基本用途**

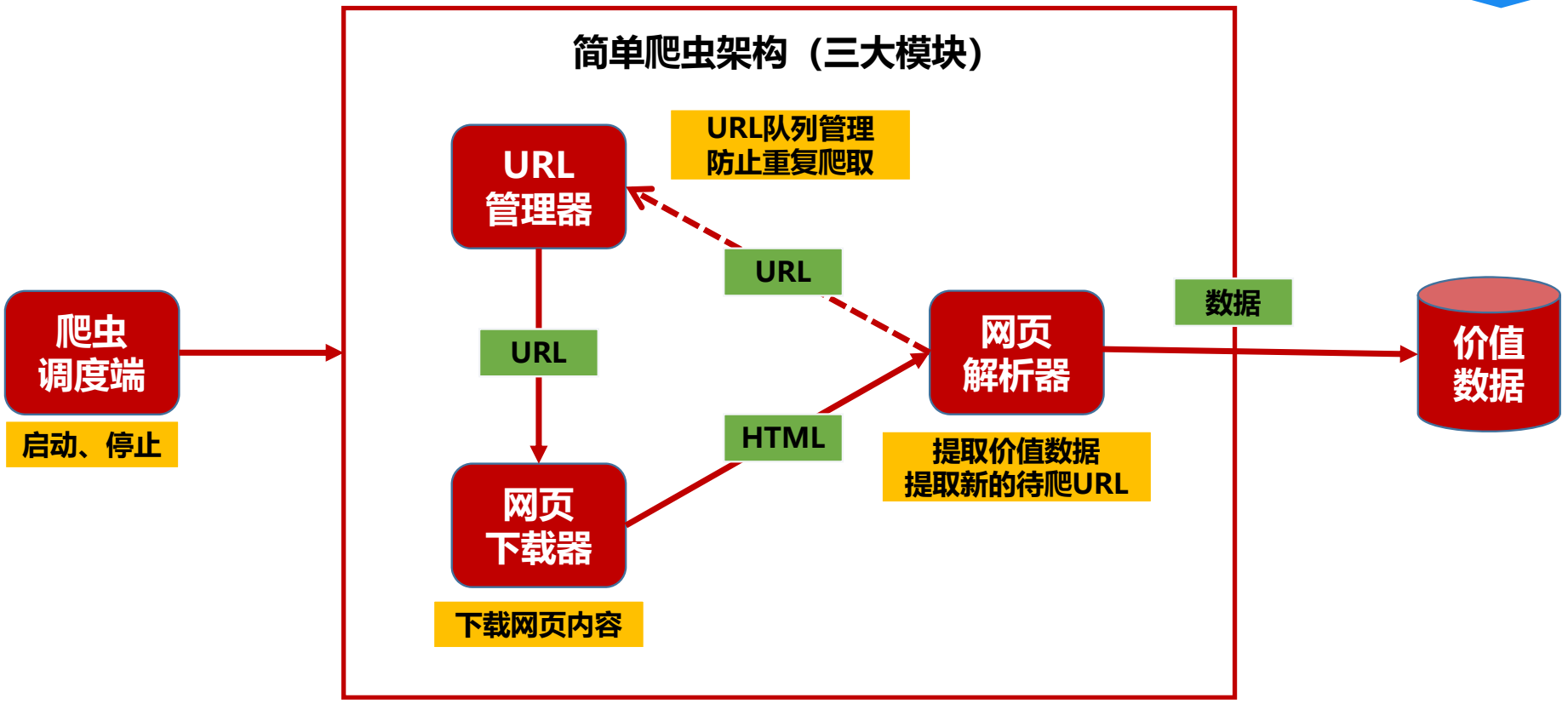
数据展示

引擎优化

特定应用

数据分析

- **爬虫的基本流程**



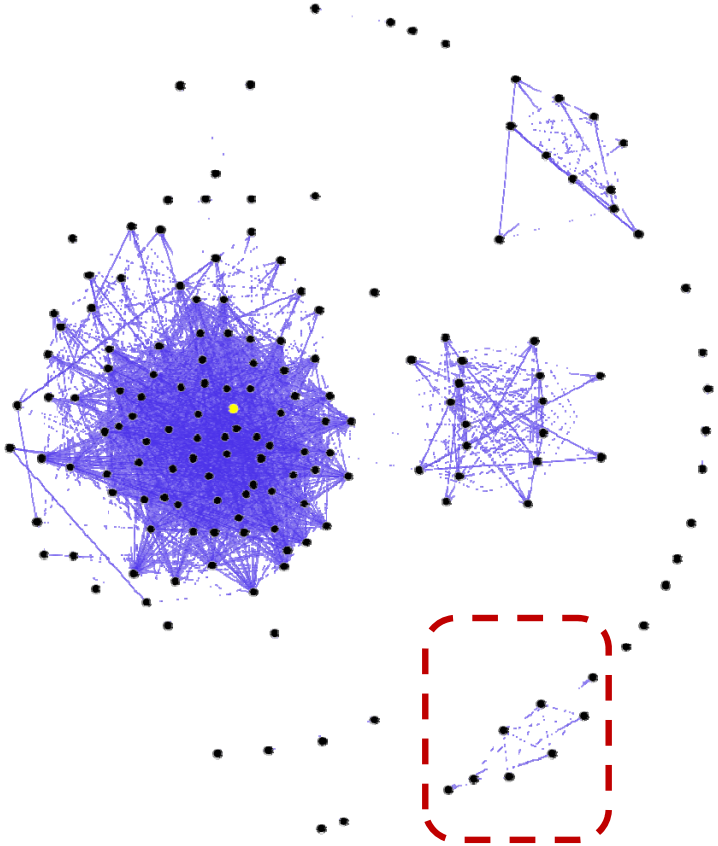
从爬取网页中获取更多URL，充实URL库，进而获取新的网页

- **网络爬虫的性能衡量**

- 数量覆盖率：“全”
  - 搜索引擎索引的网页，占目标区域中所有可能网页数量的百分比
- 质量覆盖率：“好”
  - 搜索引擎索引的网页中，“高质量”网页占目标区域中所有重要网页的百分比

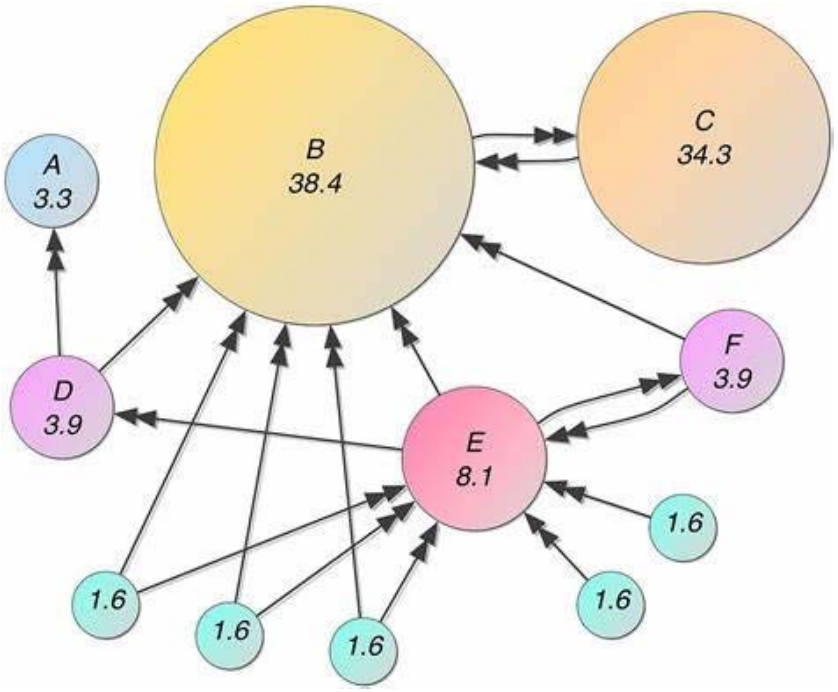
- **爬虫的数量覆盖率问题——“全”**

- 遍历完备性无法保证
  - 孤立节点的存在（如右图右半部分）
- 其他影响遍历的因素
  - 部分“偏远”节点难以遍历
  - 网络结构的动态演化
  - IP/Robots等条件的约束



- **爬虫的质量覆盖率问题——“好”**

- 如何度量网页的重要性?
  - 启发式方法：出入度…
  - 基于结构：PageRank/HITS…
  - 开放问题：信息密度衡量?
- 不仅重要，还要保证时效
  - “时新性”的要求



- **爬虫的主要需求**

- 速度（Speed）：突破网络瓶颈，最短时间内获取所需网页
- 可扩展性（Scalability）：基于多爬虫机制，有效打破单爬虫效率天花板
- 友好性（Politeness）：遵循网络秩序，不影响网络服务正常运转
- 健壮性（Robustness）：有效应对服务器陷阱等潜在问题
- 时新性（Freshness）：保持内容的时效性，提升用户体验

- **爬虫需求：速度**

- 2012年的统计数据：谷歌每天需要爬取200亿个页面
  - 以每周1400亿个页面计算，约为2^{37}个页面
  - 平均每个页面64KB，约为2^{16}个bit
    - 实际情况远远不止64KB，大量非文本内容
  - 可知每秒流量>100Gb/s
- 网络瓶颈的存在对速度的限制
  - 以家用100Mbps网络为例，考虑以太网MTU为1500字节，即使达到100%利用率，每秒也仅传输约8333个网页。

- **爬虫需求：可扩展性**

- 单一爬虫工作效率低下，难以突破效率天花板。
- 采用多爬虫机制已成为大型搜索引擎的工作常态。
- 多个爬虫带来新的挑战
  - 如何管理多个并发的连接?
  - 不同爬虫负责不同URL，如何进行划分?
  - 过多的硬件并行好处并不大
    - 抓取的性能瓶颈出现在通讯和硬盘读写

- **爬虫需求：友好性**

- 不能显著影响被爬取的服务器性能
  - 大量DNS查询可能造成类似DOS的效果
- 有些服务器可能不希望被爬取
  - Robots exclusion
  - 反面教材：360搜索、头条搜索

- **爬虫需求：健壮性**

- 如何有效应对爬取过程中所面临的各种风险？
  - 爬取网页时陷入回路怎么处理？
  - URL不规范如何解决？
  - 服务器陷阱如何应对？
  - 系统崩溃如何处理？

- **爬虫需求：健壮性（续）**

- 一些常见的服务器陷阱
  - 病态HTML文件
    - 例如，包含大量null字符的网页
  - 误导爬虫的网站
    - CGI程序生成的无限个网页
    - 自动创建很深的路径
    - HTTP服务器中的路径重映射

- **www.troutbums.com/Flyfactory/hatchline/hatchline/hatchline/flyfactory/flyfactory/flyfactory/flyfactory/flyfactory/flyfactory/flyfactory/flyfactory/hatchline**

- **爬虫需求：时新性**

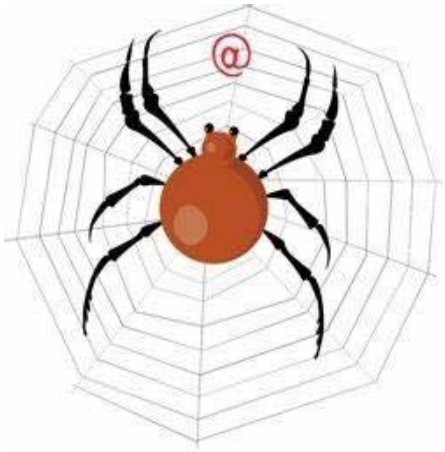
- T时刻的时新性：所抓取的网页内容与T时刻该网页的最新内容一致
- 网页年龄：距离网页最近一次更新的时间
  - 通过对网页更新行为的建模来预测年龄
- 保持爬取内容的时新性能够提升搜索效果
  - 但同时，过度重视时新性将严重增加搜索引擎的负担。
  - 部分网页频繁更新，增加抓取难度

本节目录

- 网络爬虫的定义与需求
- **爬虫的基本要素**
- 面向API的新爬虫任务
- 常见的爬虫算法
- 常见反爬虫机制与应对策略
- 分布式爬虫简述

- **爬虫所涉及的协议和要素**

- HTML / HTTP
- DNS / URL
- Sitemap
- Robots Exclusion



HTML

- 如何表示与获取网页中的链接结构?

- **HTML文件示例**

```
<nav class="main-menu">
  <div class="container" style="z-index:10">
    <ul class="menu">
      <li>
        <a href="http://www.ustc.edu.cn/2062/list.htm">学校概况</a>
        <ul>
          <li><a href="http://www.ustc.edu.cn/2072/list.htm">学校简介</a></li>
          <li><a href="https://www.ustc.edu.cn/2017/0328/c2071a179625/page.htm">现任领导</a></li>
          <li><a href="http://www.ustc.edu.cn/2070/list.htm">管理机构</a></li>
          <li><a href="http://www.ustc.edu.cn/2069/list.htm">校园地图</a></li>
          <li><a href="http://www.ustc.edu.cn/2068/list.htm">服务指南</a></li>
        </ul>
      </li>
      <li>
        <a href="http://www.ustc.edu.cn/2018/0622/c2061a266526/page.htm">院系介绍</a>
        <ul style="display:none;">
          <li><a href="https://scgy.ustc.edu.cn/" target="_blank">少年班学院</a></li>
          <li><a href="http://math.ustc.edu.cn/" target="_blank">数学科学学院</a></li>
          <li><a href="http://physics.ustc.edu.cn/" target="_blank">物理学院</a></li>
          <li><a href="https://scms.ustc.edu.cn/" target="_blank">化学与材料科学学院</a></li>
          <li><a href="/_s125/main.htm" target="_blank">生命科学学院</a></li>
          <li><a href="/_s39/main.htm" target="_blank">工程科学学院</a></li>
          <li><a href="/_s93/main.htm" target="_blank">信息科学技术学院</a></li>
          <li><a href="/_s56/main.htm" target="_blank">计算机科学与技术学院</a></li>
          <li><a href="/_s75/main.htm" target="_blank">地球和空间科学学院</a></li>
          <li><a href="/_s45/main.htm" target="_blank">管理学院</a></li>
```

- **HTML的基本概念**

- HyperText Markup Language，书写网页的“框架语言”
- 基本组成：“标记”（Tags）+“文本内容”（Text）
  - 例如：学校概况
- 标记的作用：
  - 说明网页元数据（如“标题”等）
  - 说明文本内容的布局和字体、字号等信息
  - 嵌入图片、视频、创建超链接等

- **HTML的示例框架**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>
        <title> This is the title but often omitted </title>
</head>
<body>
        <img src = “url1” alt=“text”>
        other text
        <a href = “url2” title=“anchor text”> this is link text </a>
</body>
</html>
```

HTML

- **HTML中具有特别意义的文字**

- <head><title> text </text></head>
- 是搜索服务显示的内容之一（URL，标题，摘要等）
- <img src="url" alt="text">
- 图片描述，可以帮助我们做“从文字到图片”的查询
- link text
- 有助于理解目标网页内容及网页之间在内容上的关系

HTTP

- **HTTP的基本概念**

- 超文本传输协议（HyperText Transport Protocol）
  - 工作在TCP 之上（请求/应答方式）
  - 容许在一个TCP 连接上发多个HTTP请求
- 工作步骤（从客户端看）
  - 通过域名服务器（DNS ）得到服务器主机的IP地址
  - 用TCP和服务器建立联系
    - 发送HTTP 请求（例如，GET或POST）
    - 接收HTTP应答头
    - 接收HTML网页内容

- **HTTP的应用实例**

User agent's request

Server's response

HTML code of returned webpage

```
telnet www.ics.uci.edu 80
Trying 128.195.1.77...
Connected to lolth.ics.uci.edu.
Escape character is '^]'.
GET http://www.ics.uci.edu/ HTTP/1.1
Host: www.ics.uci.edu

HTTP/1.1 200 OK
Date: Wed, 25 Sep 2002 19:43:12 GMT
Server: Apache/1.3.26 (Unix) PHP/4.1.2 mod_ssl/2.8.10 OpenSSL/0.9.6e
X-Powered-By: PHP/4.1.2
Transfer-Encoding: chunked
Content-Type: text/html

f00
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Information and Computer Science at the University of
California, Irvine</title>
...
```

Refer to RFC 2616

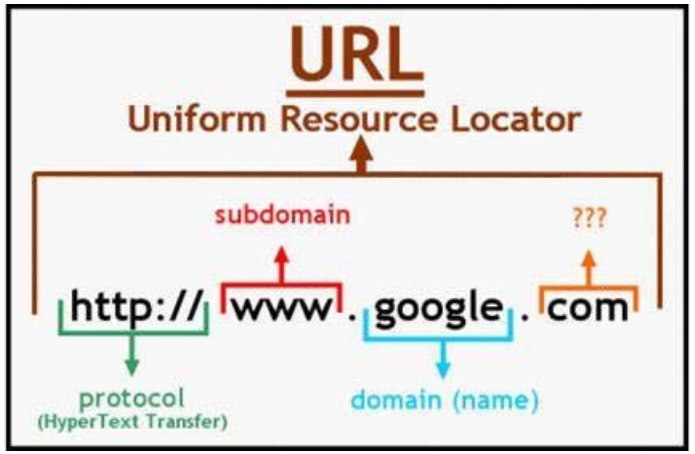
Example of the use of the GET method in an HTTP 1.1 session.

URL

- **URL的基本概念**

- 统一资源定位符（Universal Resource Locator）
  - 以某种统一的（标准化的）方式标识资源的简单字符串。

- URL一般由四部分组成：
  - 访问资源的模式/协议。
  - 存放资源的主机名。
  - 资源自身的名称，由路径表示。
  - 被访问的文件名或主页名。



- **URL实例**

- URL例子：http://cs.ustc.edu.cn/3058/list.htm

  - 这是一个可通过HTTP协议获得的文档。
  - 存放在名为cs.ustc.edu.cn的主机名。
  - 通过路径3058（对应学院新闻栏目）访问。

URL与IP地址的对应关系

- 一般而言，域名与IP地址处于一一对应的关系
- 实际情况下，域名与IP地址之间存在复杂的对应关系
  - 一对一：基本的对应关系
  - 一对多：可能由于虚拟主机导致，使得多个URL映射到同一IP
    - 例如，www.pku.edu.cn / www.gh.pku.edu.cn 等都映射到162.105.129.12上，但由于各个站点内容不同，所以被认为是不同的Web服务器
  - 多对一：可能由于DNS轮转导致，以应对商业站点的负载问题
    - 例如，多个182.61.200.x均与www.baidu.com 映射。

- **URL链接提取与规范化**

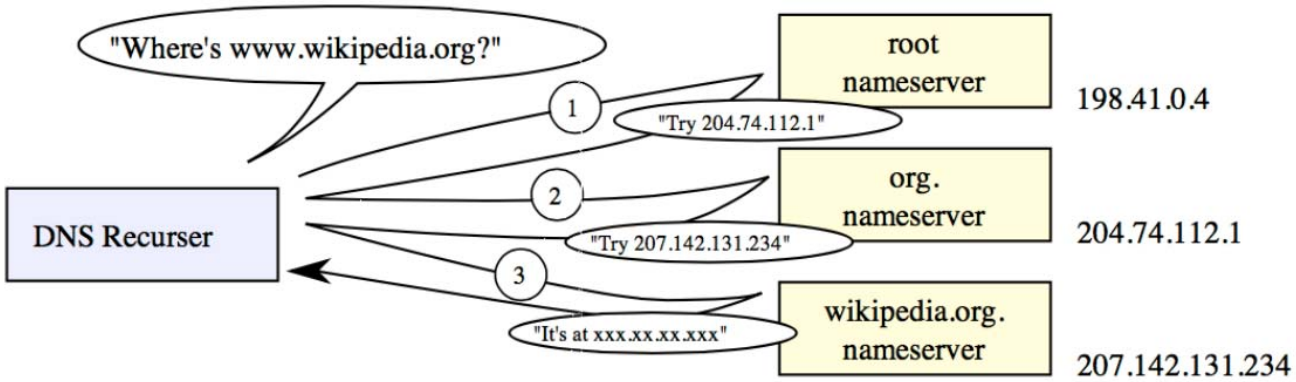
- 目标：得到网页中所含URL的标准型
  - URL的处理与过滤，避免多次获得不同URL指向的相同网页
  - 相对URL：基础URL被省略，需要进行补齐
- URL的不规范现象
  - 不同URL可能指向同一个网页
  - URL的格式存在错误或无用内容
    - 多余的文件名（如index.html），无用的查询变量或空的查询条件
  - 动态网页与动态参数
  - **短链接**

- **一些URL规范化的操作**

- URL的基本组成：协议:// 主机名[: 端口]/ 路径/[: 参数] [? 查询]#Fragment
  - protocol ://hostname[:port]/path/ [;parameters][?query]#fragment
- 一些常见的URL规范化示例
  - URL协议名和主机名的小写化
    - HTTP://WWW.BAIDU.COM -> http://www.baidu.com
  - 删除Fragment（#）或多余的查询串，如?，&
    - http://www.example.com#seo -> http://www.example.com
  - 删除默认后缀或多余的www
    - http://www.example.com/index.html -> http://example.com

- **DNS的基本概念和作用**

- 域名系统（Domain Name System）
  - 将域名和IP地址相互映射的一个分布式数据库。
  - DNS地址解析可能成为重要瓶颈，甚至造成类似DOS的攻击效果



- **提升DNS性能的方法**

- 如何提高DNS解析模块的性能?
  - 并行 DNS Client
  - 缓存 cache DNS results
  - 预取 prefetch client

• 并行 DNS Client

- 并行的地址解析Client
  - 专门对付多个请求的并行处理
  - 容许一次发出多个解析请求
  - 协助在多个DNS server之间做负载分配
    - 例如，根据掌握的URL进行适当调度

• 缓存 cache DNS results

- 增加DNS缓存的重要性
  - 面向海量URL和主机的搜索任务，如果没有DNS缓存，会造成频繁查询DNS服务器，从而造成类似于拒绝服务攻击（DOS）的副作用。
  - 针对小规模的网页搜索（如百万量级），可利用建立在内存中的DNS映射，既能加快网页信息的获取，后能降低对于DNS服务器的压力。
- DNS缓存的内容
  - Internet的DNS系统会定期刷新，交换更新的域名和IP信息
  - 缓存中有部分信息过期影响不大，但注意要适度刷新

- **预取 prefetch client**

- 为了减少等待查找涉及新主机的地址的时间，尽早将主机名投给DNS系统
- 与缓存系统的区别：缓存 – 用了才缓存；预取 – 不用也暂时先缓存
- DNS预取的基本步骤
  - 分析刚刚得到的网页
  - 从<href>属性中提取主机名
  - 向缓存服务器提交DNS解析请求
  - 结果存放在DNS缓存中（也可能用上，也可能用不上）

Sitemap

- **Sitemap，站点地图**

- 放在服务器根目录中的sitemap.xml，为爬虫指明抓取的建议
  - 协助爬虫找到隐藏的网页（如需要查询或表单才能够访问的内容）
  - Changefreq属性还提供有关更新频率的说明，以协助保障时新性

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.company.com/</loc>
    <lastmod>2008-01-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
  <url>
    <loc>http://www.company.com/items?item=truck</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.company.com/items?item=bicycle</loc>
    <changefreq>daily</changefreq>
  </url>
</urlset>
```

Robots

- **Robots Exclusion，排斥协议**

- Sitemap是允许协议，而Robots是排斥协议
- 在服务器文档根目录中的文件robots.txt，包含一个路径前缀表，描述了服务器给出的抓取限制
- 例如，腾讯新闻的Robots.txt（news.qq.com/robots.txt）

```
User-agent: *
Disallow:
Sitemap: //www.qq.com/sitemap_index.xml
Sitemap: http://news.qq.com/topic_sitemap.xml
```

Robots

- **Robots Exclusion，排斥协议**

- 中央政府门户网站的Robots.txt（www.gov.cn/robots.txt）

```
User-agent: *
Allow: /1
Sitemap:http://www.gov.cn/baidu.xml
Sitemap:http://www.gov.cn/google.xml
Sitemap:http://www.gov.cn/bing.xml
Sitemap:http://www.gov.cn/sogou.xml
Disallow:/2016gov/
Disallow:/2016shuju/
Disallow:/2016guoqing/
Disallow:/2016zhengce/
Disallow:/2016hudong/
Disallow:/2016fuwu/
Disallow:/2016xinwen/
Disallow:/premier/
Disallow:/2016guowuyuan/
Disallow:/2016public/
Disallow:/2016ducha/
Disallow:/guowuyuan/yangjing/
Disallow:/guowuyuan/yj.htm
Disallow:/c16629/gwyld_yj.htm
Disallow:/guowuyuan/yj_hy.htm
Disallow:/guoqing/2013-03/16/content_2583121.htm
Disallow:/guowuyuan/2015-12/24/content_5027563.htm
```

Robots

- **Robots Exclusion，排斥协议**

- 该限制只针对爬虫，一般浏览不受影响
- 该限制属于“君子协定”，没有强制执行力
  - 但目前，绝大多数搜索引擎都遵守该限制

本节目录

- 网络爬虫的定义与需求
- 爬虫的基本要素
- **面向API的新爬虫任务**
- 常见的爬虫算法
- 常见反爬虫机制与应对策略
- 分布式爬虫简述

API

- **API的基本概念**

- 应用程序接口（Application Programming Interface）
- 2000年，Salesforce和eBay推出了自己的API，程序员可以访问并下载一些公开数据。从那时起，许多网站都提供API用于访问公共数据库。
- 通过开放的API 来吸引更多的用户和更多的创意，与具备分享、标准、去中心化、开放、模块化的Web 2.0时代相得益彰，为创造者和平台都带来价值。
- 同时，网页API也为开发人员提供了一种更友好的网络爬虫方式：
  - 直接获取资源列表，清晰、明确
  - 接收JSON或XML等格式化数据的反馈

API

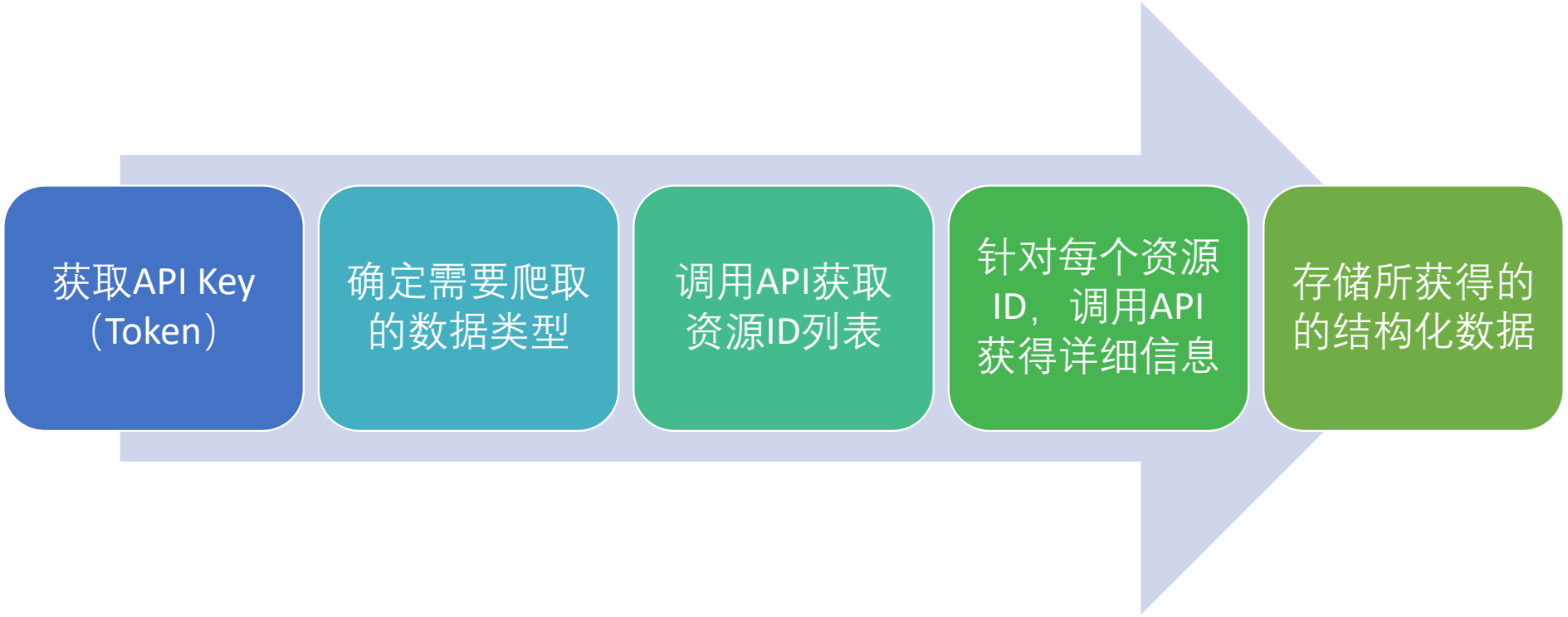
- **基于API的数据爬取示例**

```
In [28]: import urllib.request as request
         import json
         url = 'https://api.douban.com/v2/movie/top250'
         crawl_content = request.urlopen(url).read()
         top20 = json.loads(crawl_content.decode('utf8'))['subjects']
         for movie in top20:
             url = 'https://api.douban.com/v2/movie/' + movie['id']
             movieContent = request.urlopen(url).read()
             print(json.loads(movieContent.decode('utf8')))
```

```
{'rating': {'max': 10, 'average': '9.6', 'numRaters': 947593, 'min': 0}, 'author': [{'name': '弗兰克·德拉邦特 Frank Dar
abont'}], 'alt_title': '肖申克的救赎 / 月黑高飞(港)', 'image': 'https://img3.doubanio.com/view/photo/s_ratio_poster/publ
ic/p480747492.jpg', 'title': 'The Shawshank Redemption', 'summary': '20世纪40年代末，小有成就的青年银行家安迪（蒂姆·罗宾斯 Ti
m Robbins 饰）因涉嫌杀害妻子及她的情人而锒铛入狱。在这座名为肖申克的监狱内，希望似乎虚无缥缈，终身监禁的惩罚无疑注定了安迪接下来灰暗绝望的
人生。未过多久，安迪尝试接近囚犯中颇有声望的瑞德（摩根·弗里曼 Morgan Freeman 饰），请求对方帮自己搞来小锤子。以此为契机，二人逐渐熟稔，安
迪也仿佛在鱼龙混杂、罪恶横生、黑白混淆的牢狱中找到属于自己的求生之道。他利用自身的专业知识，帮助监狱管理层逃税、洗黑钱，同时凭借与瑞德的交往
在犯人中间也渐渐受到礼遇。表面看来，他已如瑞德那样对那堵高墙从憎恨转变为处之泰然，但是对自由的渴望仍促使他朝着心中的希望和目标前进。而关于其
罪行的真相，似乎更使这一切朝前推进了一步……\n本片根据著名作家斯蒂芬·金（Stephen Edwin King）的原著改编。', 'attrs': {'pubdate': ['1
994-09-10(多伦多电影节)', '1994-10-14(美国)'], 'language': ['英语'], 'title': ['The Shawshank Redemption'], 'country':
['美国'], 'writer': ['弗兰克·德拉邦特 Frank Darabont', '斯蒂芬·金 Stephen King'], 'director': ['弗兰克·德拉邦特 Frank Darab
ont'], 'cast': ['蒂姆·罗宾斯 Tim Robbins', '摩根·弗里曼 Morgan Freeman', '鲍勃·冈顿 Bob Gunton', '威廉姆·赛德勒 William Sad
ler', '克兰西·布朗 Clancy Brown', '吉尔·贝罗斯 Gil Bellows', '马克·罗斯顿 Mark Rolston', '詹姆斯·惠特摩 James Whitmore', '杰
弗里·德曼 Jeffrey DeMunn', '拉里·布兰登伯格 Larry Brandenburg', '尼尔·吉恩托利 Neil Giuntoli', '布赖恩·利比 Brian Libby',
'大卫·普罗瓦尔 David Proval', '约瑟夫·劳格诺 Joseph Ragno', '祖德·塞克利拉 Jude Ciccolella'], 'movie_duration': ['142 分
钟'], 'year': ['1994'], 'movie_type': ['犯罪', '剧情']}, 'id': 'https://api.douban.com/movie/1292052', 'mobile_link':
'https://m.douban.com/movie/subject/1292052/', 'alt': 'https://movie.douban.com/movie/1292052', 'tags': [{'count': 19
4779, 'name': '经典'}, {'count': 165432, 'name': '励志'}, {'count': 147828, 'name': '信念'}, {'count': 133384, 'name':
'自由'}, {'count': 98525, 'name': '美国'}, {'count': 94751, 'name': '人性'}, {'count': 73431, 'name': '人生'}, {'coun
t': 60696, 'name': '剧情'}]}
```

基于豆瓣提供的API，获得评分最高的20部电影信息

- **基于API的数据爬取一般流程**



其中，第3-4步可能存在循环，即通过获得某个资源ID对应的详细信息，获得更多其他资源ID，并继续抓取更多信息。

API

- **Token的概念、作用与取得方式**

- API的调用需要用户身份认证（用户授权），而Token相当于授权后的通行证
- 使用Token认证而非用户名/密码认证方式的优点
  - Token 的生成完全独立于帐号密码，往来通信不会影响账号安全
  - 即使 Token 丢失或泄露，只需登录后台删除该Token即可

目前微博的API认证授权机制

XML

- **结构化数据反馈（1）：XML**

- 可扩展标记语言（Extensible Markup Language）
  - 是一种允许用户对自己的标记语言进行定义的源语言。
  - 提供统一的方法来描述和交换独立于应用程序的结构化数据。
- XML相比于HTML的优势所在：
  - 可扩展性方面：HTML不允许用户自行定义标识或属性，而在XML中，用户能够根据需要自行定义新的标识及属性名
  - 结构性方面：HTML不支持深层的结构描述，XML的文件结构嵌套可以复杂到任意程度，能表示面向对象的等级层次。
  - 可校验性方面：HTML没有提供规范文件以支持结构校验，而XML文件可以包括一个语法描述，使应用程序可以对此进行结构校验。

- **XML文件的实例与基本要求**

```
<?xml version="1.0" encoding="UTF-8"?>
- <item>
  - <venue>
      <address_1>162 Winn St</address_1>
      <state>MA</state>
      <zip>01803</zip>
      <lat>42.504240</lat>
      <repinned>False</repinned>
      <name>American Legion Hall</name>
      <city>Burlington</city>
      <id>486621</id>
      <country>us</country>
      <lon>-71.185790</lon>
    </venue>
  - <fee>
      <label>Price</label>
      <accepts>amazon</accepts>
      <currency>USD</currency>
      <description>per person</description>
      <amount>10.0</amount>
      <required>0</required>
    </fee>
    <status>past</status>
    <description><b>Looks like the storm predicte
      confirm the band. More details will follow.
      holidays with old friends and new at a spe
      American Legion Hall in Burlington (right c
      be a cash bar. You are welcome to invite fa
      done so on your reply, it will help me keep
      on your reply, &amp;quot;paying by check&amp;quo
      soon as you know you can attend. If you h
      we had a gift exchange which was alot of f
      receiving yourself. <br />Hope it will be a
    <how_to_find_us>We have rented the hall...so
```

必须有XML声明语句作为文档开头

良好的XML文档有且只有一个根节点

例如，在本实例中为item

可根据需要嵌套文件结构

例如，<venue>中包含<city>信息

所有的标记必须有相应的结束标记

例如，<zip>与</zip>的成对出现

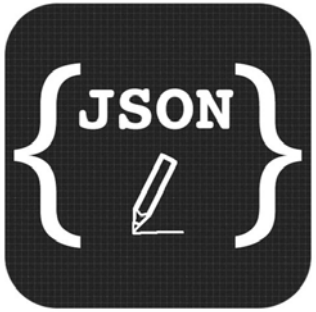
注意，所有的空标记也必须被关闭

XML

- **结构化数据反馈（2）：JSON**

- JS 对象表示法（JavaScript Object Notation）
  - 轻量级的数据交换格式，采用完全独立于编程语言的文本格式。
  - 简洁和清晰的层次结构使得 JSON 成为理想的数据存储和交换语言。
  - 易于人阅读和编写，同时也易于机器解析，并有效地提升网络传输效率。

```
"name":"Michael",
"address":
{
    "city":"Beijing",
    "street":" Chaoyang Road ",
    "postcode":100025
}
```

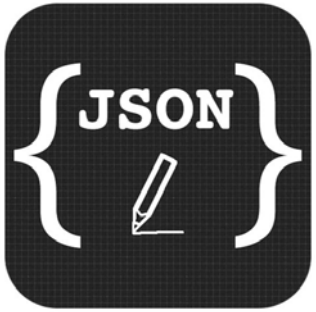


- **JSON与XML的区别**

XML

- XML的优点：格式统一、标准，容易与其他系统进行远程交互，方便共享
- XML的缺点：文件庞大，文件格式复杂，解析复杂且方式繁多，工作量大

- JSON的优点
  - 数据格式比较简单，易于读写，格式经过压缩，占用带宽小；
  - 支持多种语言，易于解析，可以简单的进行数据的读取；
  - 能直接为服务器端代码使用，大大简化了开发和维护工作。
- JSON的缺点：在结构未知的情况下解析较为困难



- **JSON的基本元素**

XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<root>
    <resultcode>200</resultcode>
    <reason>Return Successd!</reason>
    <result>
        <area>江苏省苏州市</area>
        <location>电信</location>
    </result>
</root>
```

XML与JSON
格式对比

```json
{
    "resultcode":"200",
    "reason":"Return Successd!",
    "result":{
        "area":"江苏省苏州市",
        "location":"电信"
    }
}
```

- JSON中的六大构造字符
- “[” “]”表示数组开始/结束
- “{” “}”表示对象开始/结束
- “：”表示名称的分隔符，“，”表示值的分隔符
- 从内容表示上看，JSON更为轻量，但在无缩进情况下层次化解析相对困难

本节目录

- 网络爬虫的定义与需求
- 爬虫的基本要素
- 面向API的新爬虫任务
- **常见的爬虫算法**
- 常见反爬虫机制与应对策略
- 分布式爬虫简述

- **最最基础的算法**

```
PROCEDURE SPIDER 1 (G)
        Let ROOT := any URL from G
        Initialize STACK <stack data structure>
        Let STACK := push(ROOT, STACK)
        Initialize COLLECTION <big file of URL-page pairs>

While STACK is not empty,
        URLcurr := pop(STACK)
        PAGE := look-up(URLcurr )
        STORE(<URLcurr , PAGE>, COLLECTION)
        For every URLi in PAGE,
                push(URLi , STACK)
Return COLLECTION
```

潜在风险：

重复收集的问题？

回路或不连通的解决方法？

如何控制搜集特定的一部分？

- **最最基础的改进算法**

```
PROCEDURE SPIDER(G, {SEEDS})
    Initialize COLLECTION <big file of URL-page pairs>
    Initialize VISITED <big hash-table>
    For every ROOT in SEEDS
        Initialize STACK <stack data structure>
        Let STACK := push(ROOT, STACK)
        While STACK is not empty,
            Do URLcurr := pop(STACK)
                Until URLcurr is not in COLLECTION
            insert-hash(URLcurr , VISITED)
            PAGE := look-up(URLcurr )
            STORE(<URLcurr , PAGE>, COLLECTION)
            For every URLi in PAGE,
                push(URLi , STACK)
Return COLLECTION
```

解决方案：

利用BigTable排除重复部分

基于种子集合控制爬取内容

通过更换种子重启动解决回路等

- **重复性的监测问题**

- 不仅URL本身可能重复，即使不同URL，也可能导致相似的文档内容
- 完全重复文档的检测方法
  - 检验和（Checksumming），所有字节进行加和，相同文档有相同检验和
  - 缺陷：相同检验和不代表相同（如打乱顺序），可采用循环冗余校验修正
- 近似重复文档的检测方法：指纹表示法
  - 1）对文档进行分词处理，并进行n-gram组合
  - 2）挑选部分n-gram用于表示这一文档
  - 3）对被选中的n-gram进行散列，以提升检索效率和减少指纹大小
  - 4）存储散列值作为文档指纹，通常存储在倒排索引中

- **重复性的监测问题**

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

a) 原始文本

tropical fish include, fish include fish, include fish found, fish found in, found in tropical, in tropical environments, tropical environments around, environments around the, around the world, the world including, world including both, including both freshwater, both freshwater and, freshwater and salt, and salt water, salt water species

b) 3-gram

938 664 463 822 492 798 78 969 143 236 913 908 694 553 870 779

c) 散列值

664 492 236 908

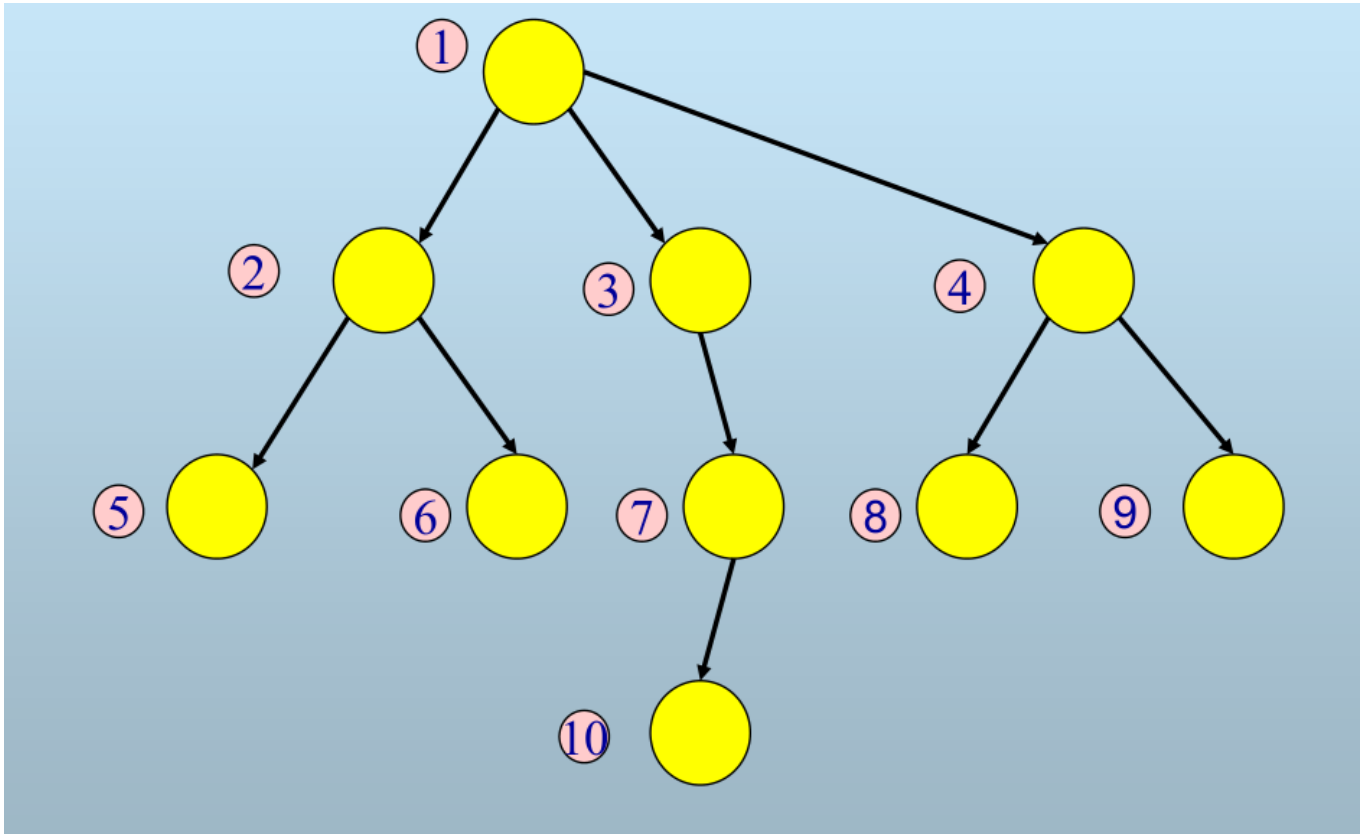
d) 使用 0 mod 4 选择的散列值

图3-14 指纹生成过程实例

- **更进一步的遍历策略，从某博士读论文开始说起**

- 遍历所有参考文献，然后从第一篇文献开始，遍历参考文献的参考文献

- **第一种遍历策略：广度优先算法**



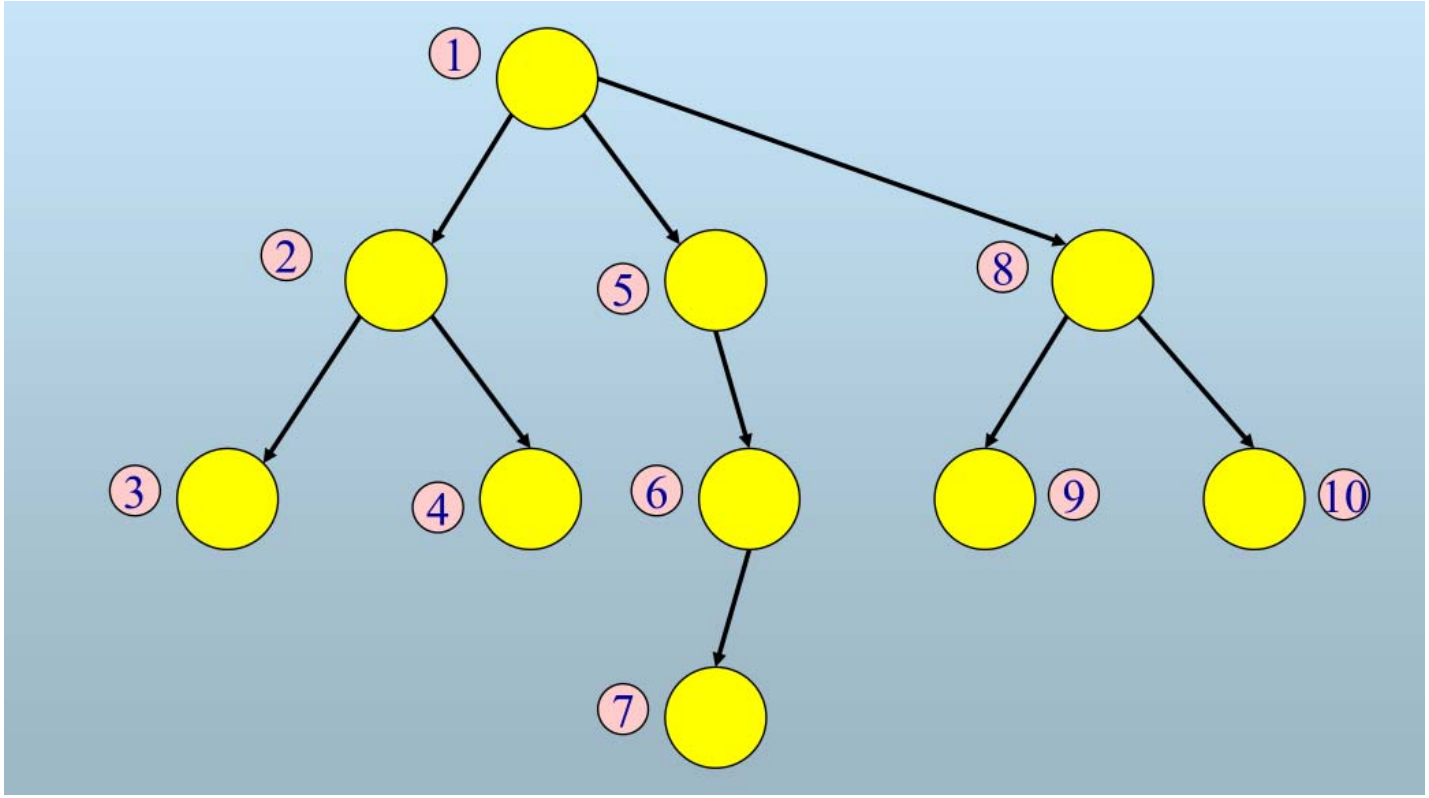
广度优先

• 第一种遍历策略：广度优先算法

```
PROCEDURE SPIDER(G, {SEEDS})
    Initialize COLLECTION <big file of URL-page pairs>// 结果存储
    Initialize VISITED <big hash-table>// 已访问URL 列表
For every ROOT in SEEDS
    Initialize QUEUE <queue data structure>// 待爬取URL 队列
    Let QUEUE := EnQueue(ROOT, QUEUE)
    While QUEUE is not empty,
            Do URLcurr := DeQueue(QUEUE)
                    Until URLcurr is not in VISITED
    insert-hash(URLcurr , VISITED)
    PAGE := look-up(URLcurr )// 爬取页面
    STORE(<URLcurr , PAGE>, COLLECTION)
    For every URL i in PAGE,// 链接提取
            EnQueue(URL, QUEUE)
Return COLLECTION
```

本质是维持一个URL队列，
先进先出

- **第二种遍历策略：深度优先算法**



深度优先

• 第二种遍历策略：深度优先算法

```
PROCEDURE SPIDER(G, {SEEDS})
    Initialize COLLECTION <big file of URL-page pairs>// 结果存储
    Initialize VISITED <big hash-table>// 已访问URL 列表
For every ROOT in SEEDS
    Initialize STACK <stack data structure>// 待爬取URL 栈
    Let STACK := push(ROOT, STACK)
    While STACK is not empty,
        Do URLcurr := pop(STACK)
            Until URLcurr is not in VISITED
        insert-hash(URLcurr , VISITED)
        PAGE := look-up(URLcurr )// 爬取页面
        STORE(<URLcurr , PAGE>, COLLECTION)
        For every URLi in PAGE,// 链接提取
            push(URLi , STACK)
Return COLLECTION
```

本质是维持一个URL栈，
先进后出

- **再进一步，从遍历要求到网页本身重要性的要求**

- 需要对网页的重要性进行评估，进而优先搜集重要的网页
- 根据经验，体现网页重要度的常见特征类别：
  - 启发式特征（简单统计指标）
    - 如，入度、父网页入度、镜像度、较小的目录深度（易于浏览）
  - 结构性特征
    - 如，PageRank/HIT值，各类Betweeness值等
  - 主题类特征（反应网页与特定需求的契合程度）

本节目录

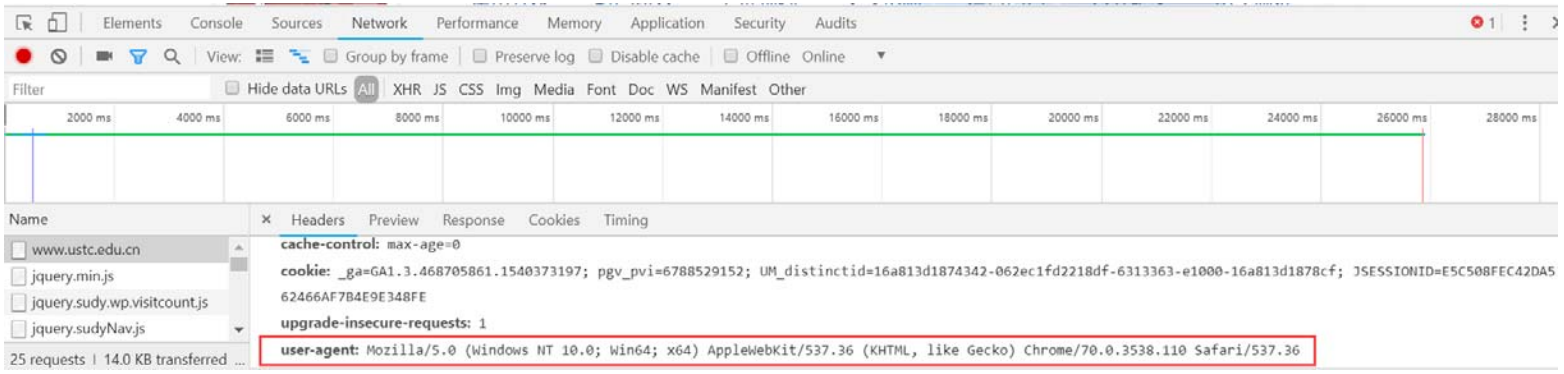
- 网络爬虫的定义与需求
- 爬虫的基本要素
- 面向API的新爬虫任务
- 常见的爬虫算法
- **常见反爬虫机制与应对策略**
- 分布式爬虫简述

- **有爬虫，就有反爬虫**

- 使用任何技术手段，阻止别人批量获取自己网站信息

- **常见反爬虫策略（1）： User Agent**

- 最常见的反爬虫策略，利用访问网站时浏览器发布的Request Headers信息中的User-Agent信息，判断用户使用何种方式浏览



UA.py

```
1  import requests
2
3  html = requests.get('https://zhihu.com').content
4  print(html.decode())
5
```

爬虫往往U-A部分为空

- **常见反爬虫策略（1）：User Agent**

- 应对策略：利用Python的Request库允许用户自定义请求头信息的手段
  - 在请求头信息中将 User-Agent 的值改为浏览器的请求头标识，从而绕开反爬虫机制

```
import requests
# 伪造请求头信息 欺骗服务器
headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:9527.0) Gecko/20100101 Firefox/9527.0"}
resp = requests.get("http://127.0.0.1", headers=headers)
print(resp.status_code)
```

- **常见反爬虫策略（2）：IP/账号访问次数/频率**

- 通过限制特定IP地址/账号访问频率和次数进行反爬
  - 其本质在于判断浏览行为是否是**人类行为**
- 应对手段：
  - 构造 IP 代理池，然后每次访问时随机选择代理
    - Github中有相关服务，通过各种提供免费IP的网站来提供代理池
  - 每次爬取行为后间隔一段时间
  - 注册多个账号以保障数据收集
    - 挑战：账号本身的成本问题、账号被查封的危险

- **常见反爬虫策略（3）：验证码**

- 通过各类验证码，判断浏览者属于人类还是机器

从简单的字符识别到
复杂的逻辑推理

- 应对手段：
  - 简单的字符识别：基于机器学习与模式识别相关技术
  - 复杂的逻辑推理：人工辅助破解

- **常见反爬虫策略（4）：动态网页**

- 从网页的 url 加载网页的源代码之后，会在浏览器里执行JavaScript程序
  - 网页内容由脚本加载，而直接抓取则只能得到空白页面

- 应对手段：
  - 核心思路：模拟调用请求
  - 使用审查元素分析ajax请求，如此循环直到获得包含数据信息的json文件

- **常见反爬虫策略（5）：蜜罐技术**

- 网页上会故意留下一些人类看不到或者绝对不会点击的链接。由于爬虫会从源代码中获取内容，所以爬虫可能会访问这样的链接
- 只要网站发现有IP访问这个链接，立刻永久封禁访问者，从而难以继续爬取

- 应对手段：
  - 核心思路：干涉爬虫路径
  - 通过工具库判断页面上的隐含元素，使爬虫避开这些元素，可以部分回避蜜罐的诱导。

- **常见反爬虫策略（6）：用户权限限制**

- 不同类型/级别的用户给予不同的内容权限
  - VIP、SVIP、蓝钻、红钻、绿钻、各种钻……

- 应对手段：
  - 氪金，就可以变强

- **其他的反爬虫策略**

- 不同的网页结构：每个相同类型的页面的源代码格式均不相同
- 双刃剑：增加爬取难度的同时降低用户浏览体验

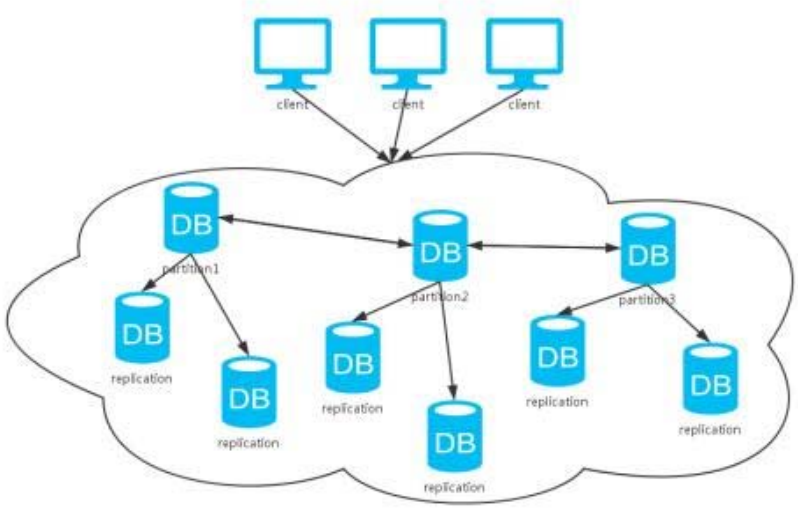
- 多模态的呈现方式：文字转为图像或视频
- 应对策略：OCR、语音识别、图像/视频标签技术

本节目录

- 网络爬虫的定义与需求
- 爬虫的基本要素
- 面向API的新爬虫任务
- 常见的爬虫算法
- 常见反爬虫机制与应对策略
- **分布式爬虫简述**

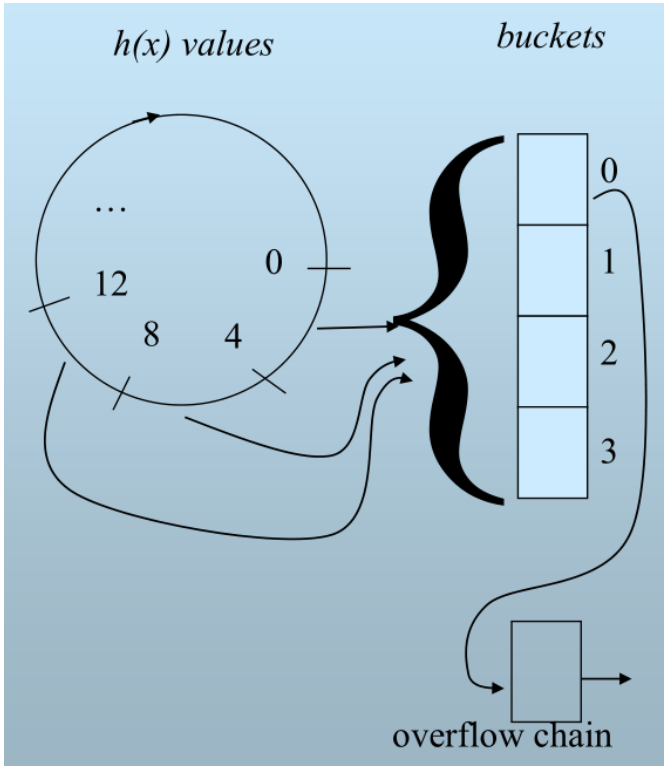
- **分布式爬虫的基本任务**

- M个节点同时执行搜集
- 问题：如何有效的把N个网站的搜集任务分配到M个机器上去？
- 目标：任务分配得均匀（Balance）



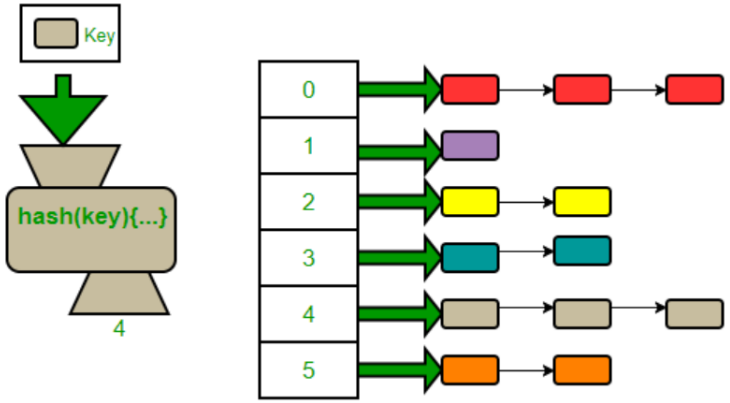
- **如何实现均匀分配**

- 首先，需要一个值均匀分布的哈希函数
  - h(name) -> 某个值（如整型数值）
- 其次，将该值映射到hash buckets
  - 对于整型数值可以简单取模（mod #buckets）
- 最后，将Item放入不同的Buckets中



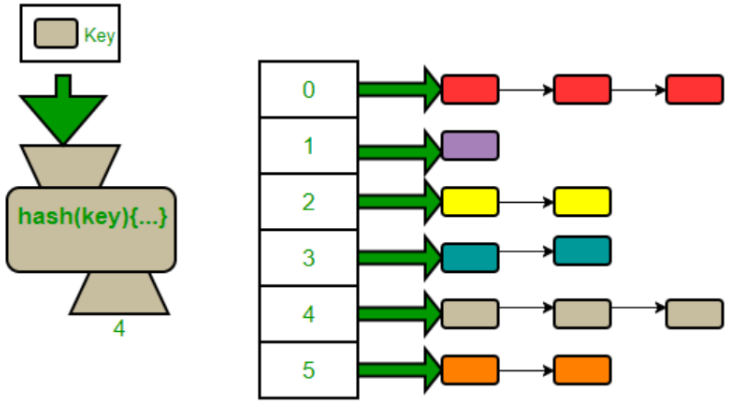
- **在机器间划分Hash Table**

- 简单划分： 把buckets按组分配到对应机器上去
- 新问题的出现
  - 节点的新增、崩溃、重新加入等特殊事件
  - 机器数目可能发生变化
  - Hash函数更改导致重新分配
  - ……

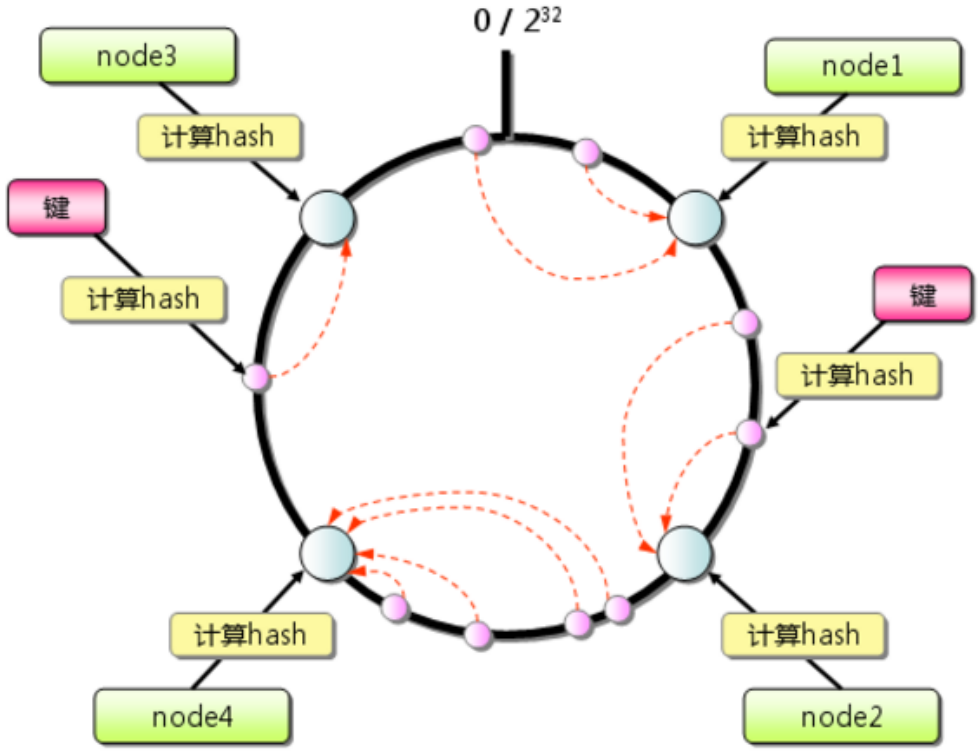


- **一致性哈希**

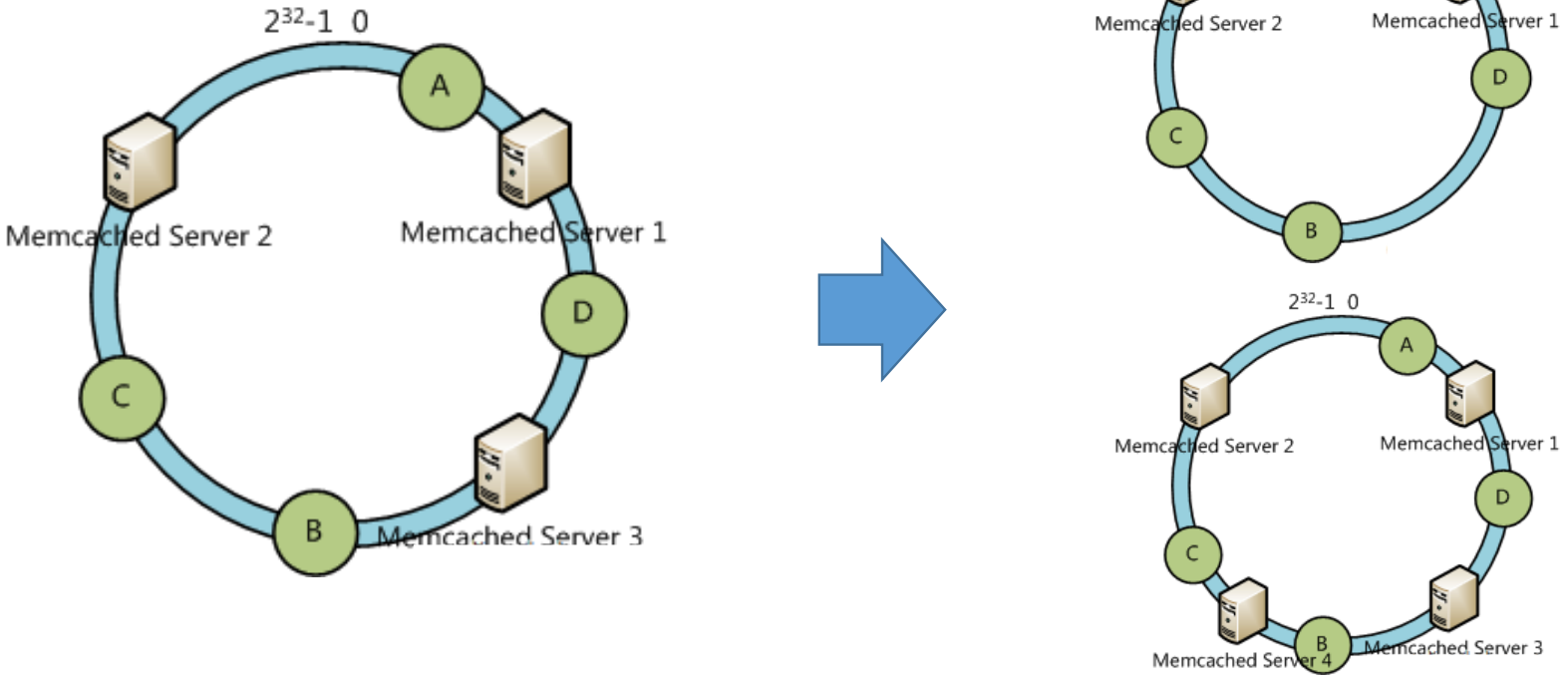
- 使用一个巨大的hash key空间，并组织成回路
- URL 和目标网站 IP 都 hash 映射到一个hash key空间，每个hash key对应一台抓取计算机
- 如果某台爬虫失效，则该机器中的URL都迁移到顺时针方向的下一个节点
  - 保证最小的数据迁移
  - 保证负载均衡，因为每个bucket都很小



- 一致性哈希


0 / 2^32
node3
计算hash
键
计算hash
node1
计算hash
键
计算hash
计算hash
node4
计算hash
node2

- **一致性哈希**
- 容错性与可扩展性分析



当增减新节点时，只有少量URL重新分配，而其他不受影响

本章小结

网络爬虫

- 网络爬虫的定义与需求
- 爬虫的基本要素
- 面向API的爬虫任务
- 常见的爬虫算法
- 常见反爬虫机制与应对策略
- 分布式爬虫简述

tongxu@ustc.edu.cn