Web Data Mining

Lecture 2: Data Access and Acquisition Methods

Milan Dojčinovski
milan.dojchinovski@fit.cvut.cz

Czech Technical University in Prague - Faculty of Information Technologies - Software and Web Engineering

Summer semester 2019/2020
Humla v0.3

Overview

- Introduction
- Web Crawler Architecture
- Crawl Control Technologies

Crawling in a Nutshell

- Automatic harvesting of web content
  - *texts, images, videos, tweets, pdfs, spreadsheets, etc.*
- Specialized intelligent programs - Web Crawlers
  - *also known as robots, bots or spiders*
- Recursive visit of web pages
  - *visit a seed web page URL, download content, extract links, add new links to the queue*
- Application of set of policies (rules)
  - *do not visit already visited pages*
  - *ignore links to images, videos or other links not pointing to a web page*

Challenges

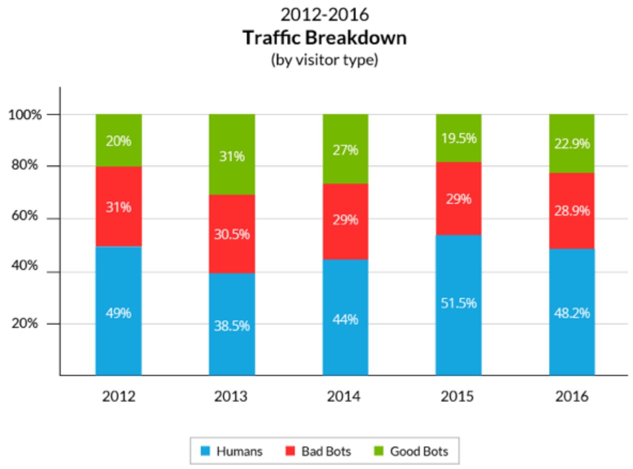
- The Web is big
  - *insufficient resources to crawl entire web*
  - *only valuable should be visited, downloaded, indexed*
- Ethical issues
  - *do not overload web servers with requests*
  - *the crawlers should identify themself*
  - *compliance with the Robot Exclusion Protocol*
- Keep crawled content fresh
  - *check content update and perform incremental updates*
- Detect "search engine spamming"
  - *tricking the ranking algorithms to achieve higher rank in search results*

Applications

- Web Search Engines
  – *Google, Bing, …*
- Web Archiving
  – *Digital preservation, archives, …*
- Vertical Search Engines
  – *Car rentals, apartments, …*
- Web Data Mining
  – *Focused crawlers, …*
- Web monitoring
- Malicious web sites detections, searching for illegal content, …
- Web site/applications testing
- Mirroring

Bot Traffic Report

- Good: 12.2% Feed fetchers, 6.6% Search engine bots, 2.9% Commercial crawlers, 1,2% Monitoring bots
- Bad: 24.3% Impersonators, 2.6% Hacker tools, 1.7% Scrapers, 0.3% Spammers
- https://www.incapsula.com/blog/bot-traffic-report-2016.html

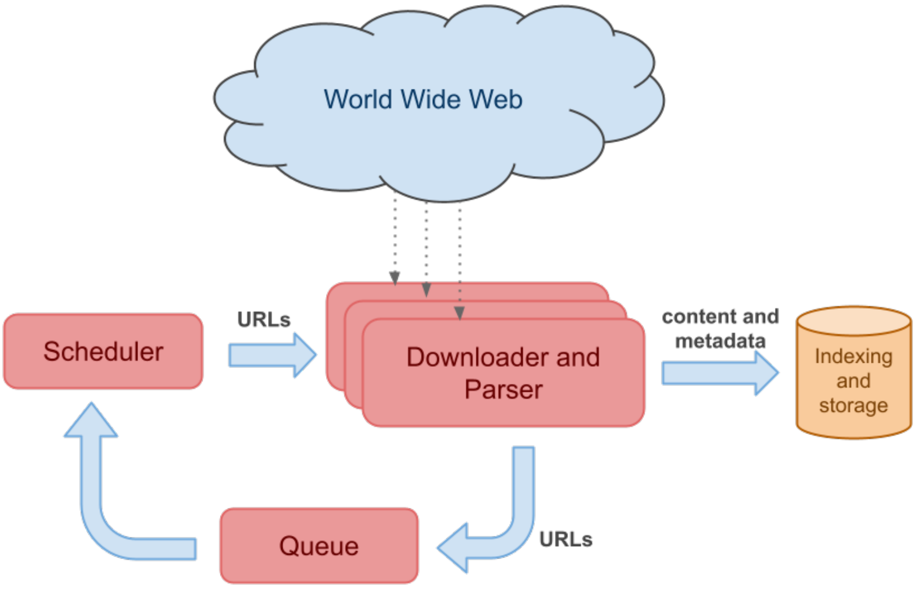


Overview

- Introduction
- Web Crawler Architecture
- Crawl Control Technologies

High-level crawler architecture

- A standard web crawler



Crawler Types

- Batch Crawlers
  - *create a snapshot of a web space*
- Incremental Crawlers
  - *continuously crawl a web space*
  - *revisit URLs to ensure freshness*
- Focused Crawlers
  - *crawl only pages in certain categories*
    → e.g. pages from only specific domain, with high popularity, specific content type, ...
- Topical Crawlers
  - *crawl only pages in certain topics*
    → e.g. sports, real estate, ...

Terminology

- Seed pages
  - *a list of URLs to visit first*
- Frontier
  - *holds a list of unvisited URLs*
- Fetcher
  - *fetches the contents of a web page*
  - *usually for the data transfer is used the HTTP protocol*
- Link Extractor
  - *parses the HTML contents and extracts URLs to other pages*
- Other Components
  - *URL filter*
    → checks whether URL is on the black-list
  - *Duplicate eliminator*
    → eliminates already visited URLs
  - *URL prioritizer*
    → assigns priority to a new URL
    → only internal(external) url, internal first, specific language

A Basic Crawling Algorithm

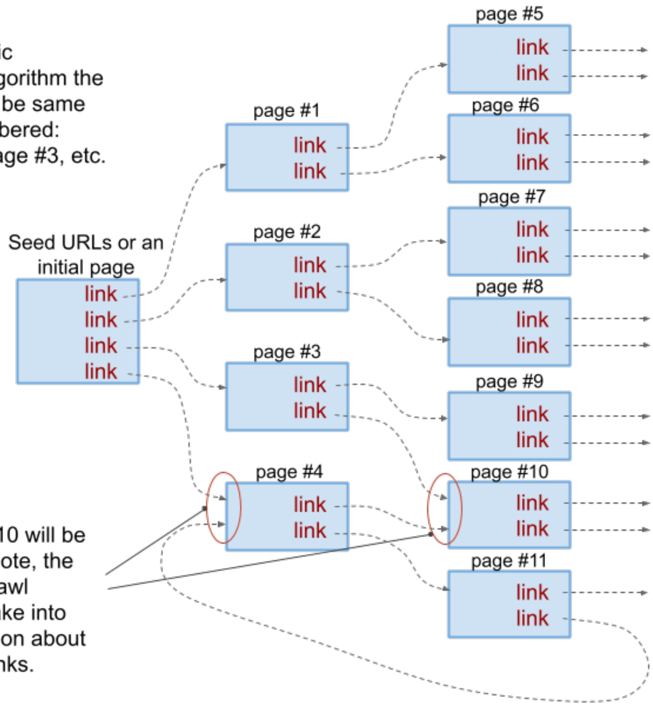
- A breadth-first crawl algorithm:
  - *Input*
    - → url_queue - *seed pages*
  - *Data structures used:*
    - → crawled_pages - *list of already visited pages*
- Steps:

```
while url_queue is not empty:
    url = get the head (first) element from the url_queue
    # The Frontier maintains the url_queue
    page = fetch the page from url
    new_links = extract links from the page's contents
    for each link in new_links:
        if link is does not exist in url_queue AND crawled_pages
            add it at the end of the url_queue list
```

- Order in which pages are visited
  - *is highly correlated with their popularity*
  - *bias of search engines to index well connected pages*

A Breadth-First Crawling

- According to the basic breadth-first crawl algorithm the order of crawling will be same as the page are numbered: page #1, page #2, page #3, etc.



- Page #4 and page #10 will be crawled only once. Note, the basic breadth-first crawl algorithm does not take into account the information about the number of backlinks.

Backlink Based Crawling Strategy

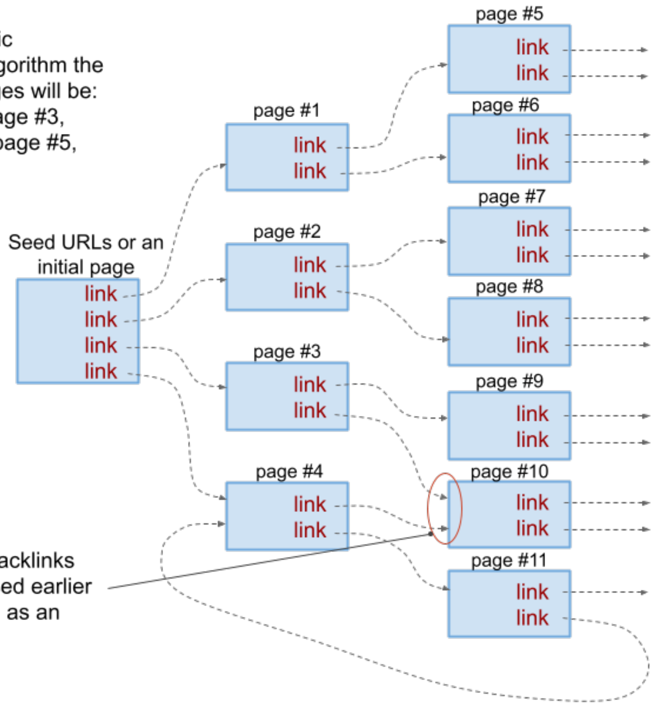
- How to select the next URL for crawling?
- Sort the URLs in the url_queue according to their relevance
  - *take into account the backlinks*
  - *backlink*
    - → *an incoming link to a web page*
- Function for sorting of URLs in the url_queue. Steps:

```
foreach url in url_queue:
    # count the number of links pointing to url
    backlink_count[url]
    sort url_queue by the backlink_count[url] information
```

- URLs with more incoming links will be processed first!
- Sorting of the url_queue is done after adding new links from a page!

Backlink Based Crawling

- According to the basic breadth-first crawl algorithm the order of crawling pages will be: page #1, page #2, page #3, page #4, page #10, page #5, page #6, ...



- Page #10 has two backlinks and it will be processed earlier since it is considered as an important page.

Fetching

- The Crawler acts as a Web client
- Sends an HTTP request to the server hosting the page and receives the response
- The crawler sends HTTP request to the server
  - *Resolving the host name in the URL to an IP address using DNS*
  - *Connecting a socket to the server and sending the request*

```
GET /wiki/Jan_Palach HTTP/1.1
Host: en.wikipedia.org
Accept: text/html
```

- The server sends response with the content of the resource http://en.wikipedia.org/wiki/Jan_Palach

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
[the other headers are not shown due to brevity]

<!DOCTYPE HTML>
<head>
    <meta charset="UTF-8" />
    <title>Jan Palach - Wikipedia, the free encyclopedia
</head>
<body>
...
</body>
```

Parsing

- Parsing the content of the HTTP payload
  - *extracting content for indexing*
  - *extracting links to be added to the frontier*
  - *extracting additional crawling and indexing directives*
  - *headers* Cache-Control, Content-Type, X-Robots-Tag, ...
- HTML code very often contains invalid markup
  - *unclosed elements, unencoded special characters, missing required attributes, improperly nested tags, missing quotes, ...*
- Bad HTML markup should be fixed
  - *a preprocessing step is required co clean up the HTML*
  - *many tools available, tidy - a tool provided by W3C*

URL Normalization

- Transform a URL into its normalized or canonical form
- URL Normalization is part of the
  - *W3C RFC 3986 specification*
- Relative URLs that are in a parsed document need normalization
  - *e.g., the website http://en.wikipedia.org/wiki/Jan_Palach contains a relative URL:* /wiki/Prague_Spring
  - *fully qualified domain name and path are not specified*
- Normalization of relative URLs
  - *relative URLs should be expanded with the fully qualified domain*
  - /wiki/Prague_Spring *becomes http://en.wikipedia.org/wiki/Prague_Spring*
  - *they do not contain the fully qualified domain name and path*

URL Normalization (cont.)

- Case Normalization
  - *the URI scheme and the protocol should be normalized to lowercase*
  - hTTp://www.EXAMPLE.com/ *is equivalent to http://www.example.com/*
  - *hexadecimal digits within a percent-encoding triplet should be normalized to uppercase (for the digits A-F)*
  - *e.g.,* "%3a" *to* "%3A"
- Percent-Encoding Normalization
  - *decoding any percent-encoded octet*
  - *http://www.example.com/%7Eusername/ becomes http://www.example.com/~username/*
  - *digits, alphabet letters, hyphens (-), period (.), underscores (_), tildes (~) should not be encoded in URIs*
- Path Segment Normalization
  - *removing dot-segments* "." *and* ".." *according to the algorithm described in RFC 3986 specs*
  - *http://example.com/a/b/../c/./ becomes http://example.com/a/c/*

URL Normalization (cont.)

- Scheme-Based Normalization
  - *may vary from scheme to scheme*
  - *for the* "http" *scheme, the default port* "80" *can be removed*
  - *http://example.com:80/ becomes http://example.com/*
- Others:
  - *Replacing IP with domain name*
  - *Removing the fragment*
  - *Sorting the query parameters*
  - *Removing unused query variables*
  - ...
- Upon performed normalization, we can use the absolute URL

Crawler Identification

- Keep track of the crawling
- Using User-Agent HTTP header
- Example:
  - User-Agent: Googlebot
  - *an user-agent string which identifies the Google's crawler*
  - *list of user-agent strings used by Google's crawlers*
- Crawler developers might consider include also additional info about the crawler
  - *e.g., URL of crawler website*

Revisit Strategies

- Pages are being added, updated or deleted continuously.
  - *Need to revisit pages to avoid outdated data.*
  - *Older a page gets, the more it costs not to crawl it.*
  - *Not possible to regularly check all pages!*
- Strategies
  - *Uniform revisiting*
    → Revisit all regularly
  - *Proportional revisiting*
    → Revisit more frequently more frequently changing pages
  - *Hybrid solution*
    → Balance resources and frequency of changes

Revisit Strategies (cont.)

- Crawlers can use HTTP HEAD to check metadata (e.g. Last-Modified, Expires headers)
  - *Freshness*
    → The proportion of pages that are fresh
    → Can lead to bad decisions, such as not crawling popular sites
  - *Age*
    → Is the # of "days" that an average page is out-of-date.

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases}$$

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at ti} \\ t - \text{mod. time of } p & \text{otherwise} \end{cases}$$

Scrapy Example

```python
import scrapy
class MySpider(scrapy.Spider):
    name = 'novinky.cz'
    start_urls = ['https://www.novinky.cz']

    def parse(self, response):
        for title in response.css('div.item > h3'):
            yield {'title': title.css('a ::text').extract_first()}

        for next_page in response.css('div.menu > ul > li'):
            yield scrapy.Request(
                response.urljoin(next_page.css(' ::attr(href)').extract_firs
                callback=self.parse)
```

```
scrapy runspider myspider.py
```

```
{'downloader/request_bytes': 7303,
 'downloader/request_count': 29,
 'downloader/request_method_count/GET': 29,
 'downloader/response_bytes': 1045117,
 'downloader/response_count': 29,
 'downloader/response_status_count/200': 28,
 'downloader/response_status_count/301': 1,
 'dupefilter/filtered': 1188,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2017, 2, 9, 14, 6, 13, 928444),
 'item_scraped_count': 196,
 'log_count/DEBUG': 227,
```

Overview

- Introduction
- Web Crawler Architecture
- Crawl Control Technologies

Robot Exclusion Protocol

- A de-facto standard defining etique policies
  - *http://www.robotstxt.org/orig.html*
  - *Consensus from 1994 between the majority of robot authors and other people with an interest in robots.*
  - *Not owned by any standardization body like W3C*
- Specifies access policies/instructions about their sites to the web robots
  - *which sites not to crawl, which bots*
- Its a simple text file: robots.txt
  - *Accessible via HTTP on the local URL* "/robots.txt"

Robots.txt Fields

- Format
  - *defined as text UTF-8 encoded file named* robots.txt
  - *must be accessible via HTTP and present in the top-level directory*
  - *one or more records separated by one or more blank lines*
  - *each record contains lines of the form:*

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

- *A record starts with one or more* User-agent *lines, followed by one or more* Disallow *lines*
- User-agent *field*
  - *the name of the robot the record is describing access policy for*
  - *if more User-agent fields are present, the same polices apply to all robots*
  - *at least one User-agent field should be present*
- Disallow *field*
  - *a partial URL that is not to be visited*
  - *any URL that starts with this value will not be retrieved*
  - *empty value indicates that all URLs can be retrieved*

Robots.txt Examples

- Allow complete access

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

- Go away!

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

- Specific robot

```
User-agent: GoogleBot
Disallow: /misc/  # Here are scripts. Do not index scripts (JavaScript).
                  # This matches /misc/script.js, but also /misc/a/script.js

# Files
Disallow: /CHANGELOG.txt    # Do not index the changelog file.

# Paths (clean URLs)
Disallow: /tmp/    # Files in /tmp/ will soon disappear.

# Paths (no clean URLs)
Disallow: /?q=logout/    # Do we really need to index the logout page?
```

- Add every URL or set of URLs that you want to exclude indexing

Robot Exclusion Protocol

- Robots.txt File Example
  – *excerpt from the robots.txt at http://fit.cvut.cz/robots.txt*

```
User-agent: *
Crawl-delay: 10
# Directories
Disallow: /includes/
Disallow: /misc/
Disallow: /modules/
# Files
Disallow: /CHANGELOG.txt
Disallow: /cron.php
Disallow: /INSTALL.mysql.txt
Disallow: /INSTALL.pgsql.txt
Disallow: /install.php
Disallow: /INSTALL.txt
# Paths (clean URLs)
Disallow: /admin/
Disallow: /comment/reply/
Disallow: /filter/tips/
Disallow: /logout/
Disallow: /node/add/
# Paths (no clean URLs)
Disallow: /?q=admin/
Disallow: /?q=comment/reply/
Disallow: /?q=filter/tips/
Disallow: /?q=logout/
```

Robots.txt Extensions

- Crawl-delay field
  - *e.g. number of seconds to wait between subsequent visits*
- Allow field
  - *specifies paths that may be accessed by the designated crawlers*
  - *useful when one tells robots to avoid an entire directory but still wants some HTML documents in that directory crawled*
  - *supported by Google, Microsoft (Bing), Yahoo*
- Sitemap field
  - *URL pointer to the Sitemap index file*
  - *the URL should be absolute!*
  - *empty value indicates that all URLs can be retrieved*
- Host field
  - *Specify preferred domain for websites with multiple mirrors*

```
User-agent: *
Allow: /
Host: www.example.com
Sitemap: http://www.example.com/sitemap.xml
```

Page-level Indexing Configuration

- Control how robots make content available through search results
  - *including a meta tag element in the HTML page, or*
  - *an HTTP response header*
- Robots meta tag

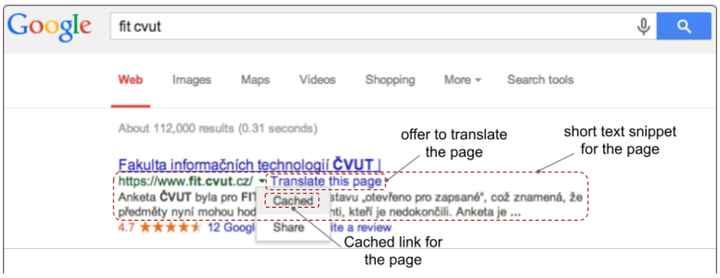
```
<!DOCTYPE html>
<html>
  <head>
    <meta name="robots" content="noindex" />
    (…)
  </head>
  <body>(…)</body>
</html>
```

- Instructs all search engines not to show the page in search

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="googlebot" content="noindex" />
    (…)
  </head>
  <body>(…)</body>
</html>
```

- Instructs Google not to show this page in its search results

Indexing and Serving Directives



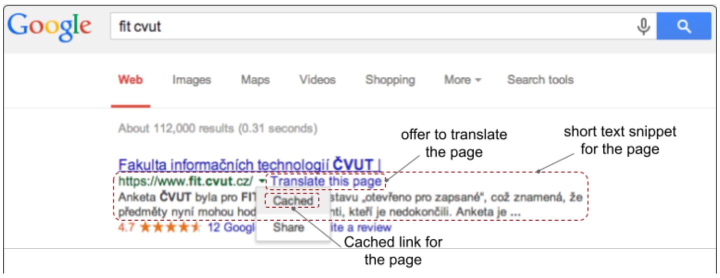
fit cvut

About 112,000 results (0.31 seconds)

Fakulta informačních technologií ČVUT |
https://www.fit.cvut.cz/ ▾ Translate this page
Anketa ČVUT byla pro FI Cached stavu „otevřeno pro zapsané", což znamená, že předměty nyní mohou hod nti, kteří je nedokončili. Anketa je ...
4.7 ★★★★★ 12 Googl Share ite a review

- noindex: do not show this page in search results, do not show cached link
- noarchive: do not show a "Cached" link in search results
- none: same as noindex, nofollow
- nosnippet: do not show a snippet in the search results for this page
- notranslate: do not offer translation of this page in search results

Indexing and Serving Directives (cont.)



fit cvut

About 112,000 results (0.31 seconds)

Fakulta informačních technologií ČVUT |
https://www.fit.cvut.cz/ ▾ Translate this page
Anketa ČVUT byla pro FI Cached stavu „otevřeno pro zapsané", což znamená, že předměty nyní mohou hod nti, kteří je nedokončili. Anketa je ...
4.7 ★★★★★ 12 Googl Share ite a review

- all: no restrictions for indexing or serving
- nofollow: do not follow the links on this page
- noimageindex: do not index images on this page
- unavailable_after: [RFC-850 date/time]: do not show this page in search results after the specified date/time
- noodp: don’t use metadata (titles or snippets) from the Open Directory project (closed on March 17, 2017)

X-Robots-Tag HTTP Header

- HTTP header, part of the HTTP response
- Any directive from the robots meta tag can be specified
- Usage example
  - *HTTP request-response*
  - X-Robots-Tag: noindex - *the server instructs the crawler not to index the page*

```
> GET /index.html HTTP/1.1
> Host: example.com
> Accept: text/html
> Date: Thu, 27 Feb 2014 10:12:77 GMT

< HTTP/1.1 200 OK
< Content-Type: text/html
< X-Robots-Tag: noindex
<
< ...resource representation in html...
```

- Bot specification
  - X-Robots-Tag: googlebot: noindex

X-Robots-Tag HTTP Header

- Multiple X-Robots-Tag headers can be specified in the HTTP response
- Multiple headers example. Specified directives:
  - *no indexing of images in the requested page*
  - *the page will not be available after* April 22nd 2014 23:59:59

```
> GET /index.html HTTP/1.1
> Host: example.com
> Accept: text/html
> Date: Thu, 27 Feb 2014 10:12:77 GMT

< HTTP/1.1 200 OK
< Content-Type: text/html
< X-Robots-Tag: noimageindex
< X-Robots-Tag: unavailable_after: 22 Apr 2014 23:59:59 PST
<
< ...resource representation in html...
```

- multiple directives can be also specified in one single header - comma-separated

```
< X-Robots-Tag: noimageindex, unavailable_after: 22 Apr 2014 23:59:59 PST
```

Web Servers Configuration for X-Robots-Tag

- Apache X-Robots-Tag header configuration
  - *Configuring Apache with* noindex *and* nofollow *directives*
  - *Do not index JavaScript files across the entire web site*

```
<Files ~ "\.js$">
  Header set X-Robots-Tag "noindex, nofollow"
</Files>
```

- Node.js X-Robots-Tag header configuration
  - *Configuring Node.js with noindex directive*

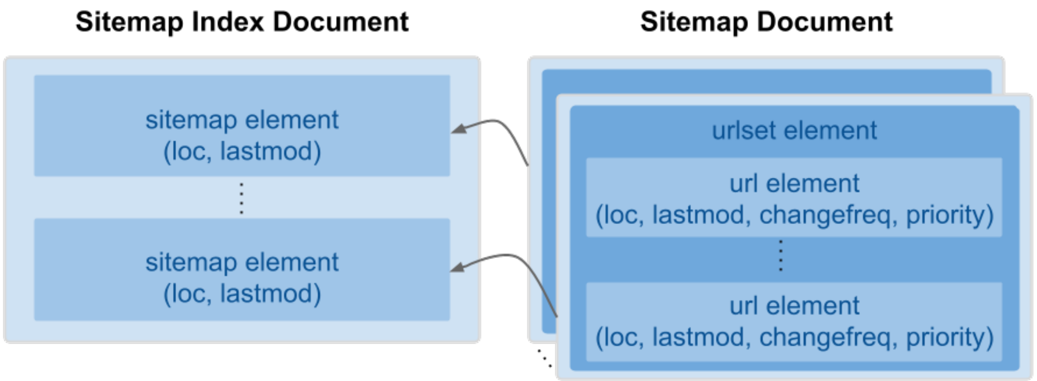
```
var http = require('http');

http.createServer(function (req, res) {
    // assuming variable 'file' holds the contents of the file
    // setting the X-Robots-Tag response header
    res.setHeader('X-Robots-Tag', 'noindex');
    res.end(file);
}).listen(8080);
```

Sitemaps

- Need for standard web site structure format
  - *Machine-processable Web site structure description*
  - *Complementary work to* robots.txt
    - → robots.txt *spec does not say how to instruct crawlers what to crawl, how often, etc.*
- Standard protocol describing web site structure
  - *https://www.sitemaps.org*
  - *URLs available for crawling*
- Introduced by Google in June 2005, sitemaps main site
- Supported by Google, Yahoo! and Microsoft
- An XML file with list of URLs and URLs metadata
- Sitemaps is a robot inclusion standard

Sitemaps Format



- Two types of sitemap documents
  - *Sitemap Index Document*
    - → *provides aggregation of multiple sitemap files, together with their location and their last modification timestamp*
  - *Sitemap Document*
    - → *represent a single sitemap document containg a list of URLs offered for crawling*

Sitemap Format

- Sitemap Index Document Example
  - *excerpt from the* sitemap.xml *at http://fit.cvut.cz/sitemap.xml*
  - *groups two sitemap files*

```
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.sitemaps.org/schemas/sitemap/0.9 http://w
    <sitemap>
        <loc>http://fit.cvut.cz/sitemap0.xml</loc>
        <lastmod>2014-02-27T15:26:35+00:00</lastmod>
    </sitemap>
    <sitemap>
        <loc>http://fit.cvut.cz/sitemap1.xml</loc>
        <lastmod>2010-02-15T15:00:56+00:00</lastmod>
    </sitemap>
</sitemapindex>
```

Sitemap Example

- Sitemap Format Example
  - *excerpt from the* sitemap0.xml *at* *http://fit.cvut.cz/sitemap0.xml*
  - *an sitemap file*
  - *groups multiple URLs*

```
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.sitemaps.org/schemas/sitemap/0.9 http://w
    <url>
        <loc>http://fit.cvut.cz/</loc>
        <lastmod>2014-02-27T15:26:35+00:00</lastmod>
        <changefreq>hourly</changefreq>
        <priority>1.0</priority>
    </url>
    <url>
        <loc>http://fit.cvut.cz/node/1</loc>
        <lastmod>2010-07-31T08:23:17+00:00</lastmod>
        <changefreq>always</changefreq>
        <priority>0.8</priority>
    </url>
    ...
</urlset>
```

Other sources of metadata

- Well-Known URIs
  - *site-wide metadata*
  - *location* /.well-known/
  - *Examples*
    - → /.well-known/host-meta
      - → *host metadata including author, copyright, etc.*
    - → /.well-known/security.txt
      - → *contacts for reporting security vulnerabilities, ...*