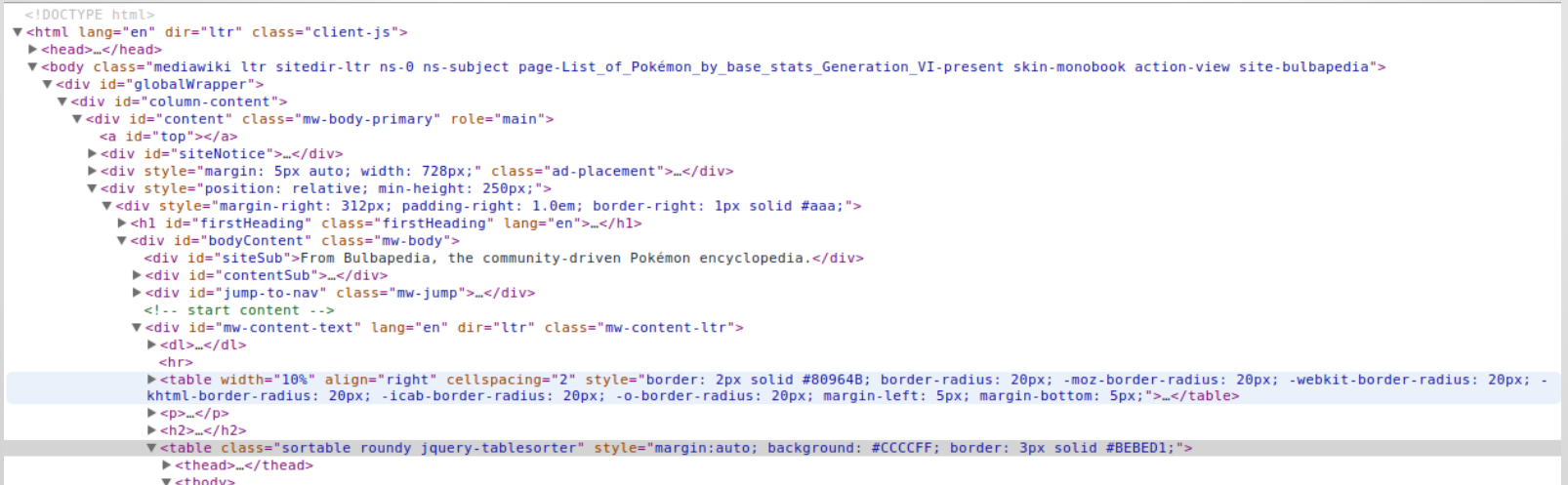
Web Scrapers/Crawlers

Aaron Neyer - 2014/02/26

Scraping the Web

- Optimal - A nice JSON API
- Most websites don’t give us this, so we need to try and pull the information out



How to scrape?

- Fetch the HTML source code
  - python: urllib
  - ruby: open-uri
- Parse it!
  - Regex/String search
  - XML Parsing
  - HTML/CSS Parsing
    - python: lxml
    - ruby: nokogiri

Examine the HTML Source

- Find the information you need on the page
- Look for identifying elements/classes/ids
- Test out finding the elements with Javascript CSS selectors

Let’s find some Pokemon!

```ruby
require 'open-uri'
require 'nokogiri'

BASE_URL = 'http://bulbapedia.bulbagarden.net'

# Fetch the pokemon list page and parse it with nokogiri
source = open("#{BASE_URL}/wiki/List_of_Pok%C3%A9mon_by_base_stats")
doc = Nokogiri::HTML(source)

# Get the rows containing the pokemon information
rows = doc.css('table.roundy tr')[1..-1]

rows.each do |row|
  # Get the number and name from the td cells content
  cells = row.children.css('td')
  number = cells[0].content.chomp
  name = cells[2].content.chomp
  # Get the URL for the pokemon's page by getting the href from the a element
  url = BASE_URL + cells[2].children.css('a')[0].attributes['href'].value
  puts "#{number} #{name} #{url}"
end
```

What about session?

- Some pages require you to be logged in
- A simple curl won’t do
- Need to maintain session
- Solution?
  - python: scrapy
  - ruby: mechanize

Want to mine some Dogecoins?

```
agent = Mechanize.new
agent.get("https://dogehouse.org/?page=login")
form = agent.page.forms[0]
form['username'], form['password'] = username, password
form.submit
agent.get("https://dogehouse.org/?page=dashboard")

rows = agent.page.parser.css('.stratumtable table').children

minimum = nil
minimum_port = nil
1.upto(rows[0].children.length-1) do |i|
  name = rows[0].children[i].content
  value = rows[1].children[i].content
  if name[0] == 'L'
    if minimum.nil? || value < minimum
      minimum = value
      minimum_port = name[2..-1]
    end
  end
end

agent.get("https://dogehouse.org/?page=gettingstarted")
urls = agent.page.parser.css('.module_content > li > table > tbody > tr kbd').map(&:content)
puts urls.find { |url| url.end_with?(minimum_port) }
```

What is a web crawler?

- A program that systematically scours the web, typically for the purpose of indexing
- Used by search engines (Googlebot)
- Known as spiders

How to build a web crawler

- Need to create an index of words => URLs
- Start with a source page and map all words on the page to it’s URL
- Find all links on the page
- Repeat for each of those URL’s
- Here is a simple example:

```ruby
until urls_to_check.empty?
  # Pop off the next url to check, and mark it as checked
  next_url = urls_to_check.pop
  urls_checked.add(next_url[:url])

  # Fetch the source code. Ignore the page if we can't fetch it correctly
  begin
    source = open(next_url[:url])
  rescue OpenURI::HTTPError, Zlib::DataError
    next
  end

  # Parse the HTML source and fetch all of the words on the page and the links
  doc = Nokogiri::HTML(source)
  links = doc.css('a')
  words_on_page = doc.content.split

  # Add all the words on the page to the index
  words_on_page.each do |word|
    index[word] ||= {}
    index[word][next_url[:url]] ||= 0
    index[word][next_url[:url]] += 1
  end

  # Get the URL's from all of the a tags that have an href
  urls = links.map{ |a| a.attributes['href'] }.compact.map{ |href| href.value }

  # Add all of the new url's to our list of pages to check if we haven't checked them already
  urls.each do |url|
    # If we have a relative url '/search' instead of 'http://google.com/search',
    # then append it to our base path
    if !url.start_with?("http")
      url = (URI.parse(next_url[:url]) + url).to_s
    end

    # If we haven't checked the URL and we aren't at our max depth, add it to our list
    if next_url[:depth] != max_depth && !urls_checked.include?(url)
      urls_to_check << {url: url, depth: next_url[:depth]+1}
    end
  end
end
```

Some improvements

- Handle URL’s better
- Better content extraction
- Better ranking of pages
- Multithreading for faster crawling
- Run constantly, updating index
- More efficient storage of index
- Use sitemaps for sources

Useful Links

- Nokogiri: http://nokogiri.org/
- lxml: http://lxml.de/
- Mechanize: http://docs.seattlerb.org/mechanize/
- Scrapy: http://scrapy.org/
- HacSoc talks: http://hacsoc.org/talks/

Any Questions?