Web Scraping with R

What is web scraping

"Web scraping" is an umbrella term for various methods for collecting data from websites. These methods can either be manual or more or less automated.

Typically "web scraping" refers to techniques that gathers information programatically from the internet in some way (i.e. not via a browser).

Working with web scraping involves both collecting the raw data from the internet as well as handling and converting these data to a data format that can be processed in some way.

Why is web scraping relevant?

1. The internet is in itself a relevant field of research

The internet is completely intertwined with our daily lives and practices. Both information on the internet and information about our use of the internet provide insights into habits, consumption and interaction.

2. The internet is a data source

The internet is a massive collection of information and we need techniques for collecting data from it.

Types of web scraping

One can discern between these main types of web-scraping:

- "Raw" web scraping: Techniques collecting the raw information (source codes) from webpages.
- API-based: Use of avilable API's (application programming interfaces) to access data made available

Within "raw" web scraping, these terms are used to describe specific kinds of scraping tools:

- Spiders: A scraper that collects information about how websites refer to each other (a network)
- Crawlers: A scraper that jumps from website to website and completes specific scraping tasks

In this workshop, you will be introduced to raw web scraping techniques.

A note on the legality of web scraping

As this workshop introduces raw web scraping tools, you are given the tools to start collecting data yourselves from various websites.

Make sure that you are allowed to scrape a website before scraping it as your scraping could be considered a criminal act and/or in violation of copyright law or personal data laws (GDPR)

Web scraping and copyright law (terms of service/use)

The data that various companies make available on websites is typically owned by those companies. Many websites have specific "terms of service" or "terms of use" in which they may specify that scraping their websites is a violation of their terms.

Companies write this as part of their terms because scraping can be used to create websites that copies content from other websites automatically.

Web scraping and personal data

Some data on the web (especially data on social media) is a huge gray area regarding personal data: Is data that a person voluntarily uploads to a public social media page still their data and personal? Some cases "yes", some cases "no", some cases hard to tell.

Also keep in mind that websites like Facebook, Twitter and Instagram have specific terms for using their data, as they are responsible for and, in principle, own the data being uploaded. Facebook and Instagram fx does not allow the automatic collection of their data (specified in their terms of service).

Web scraping and "hacking"

A website is located on a server. Every time a website is visited, the server receives a "request" that has to be processed by the server. The more requests, the busier and more congested the server.

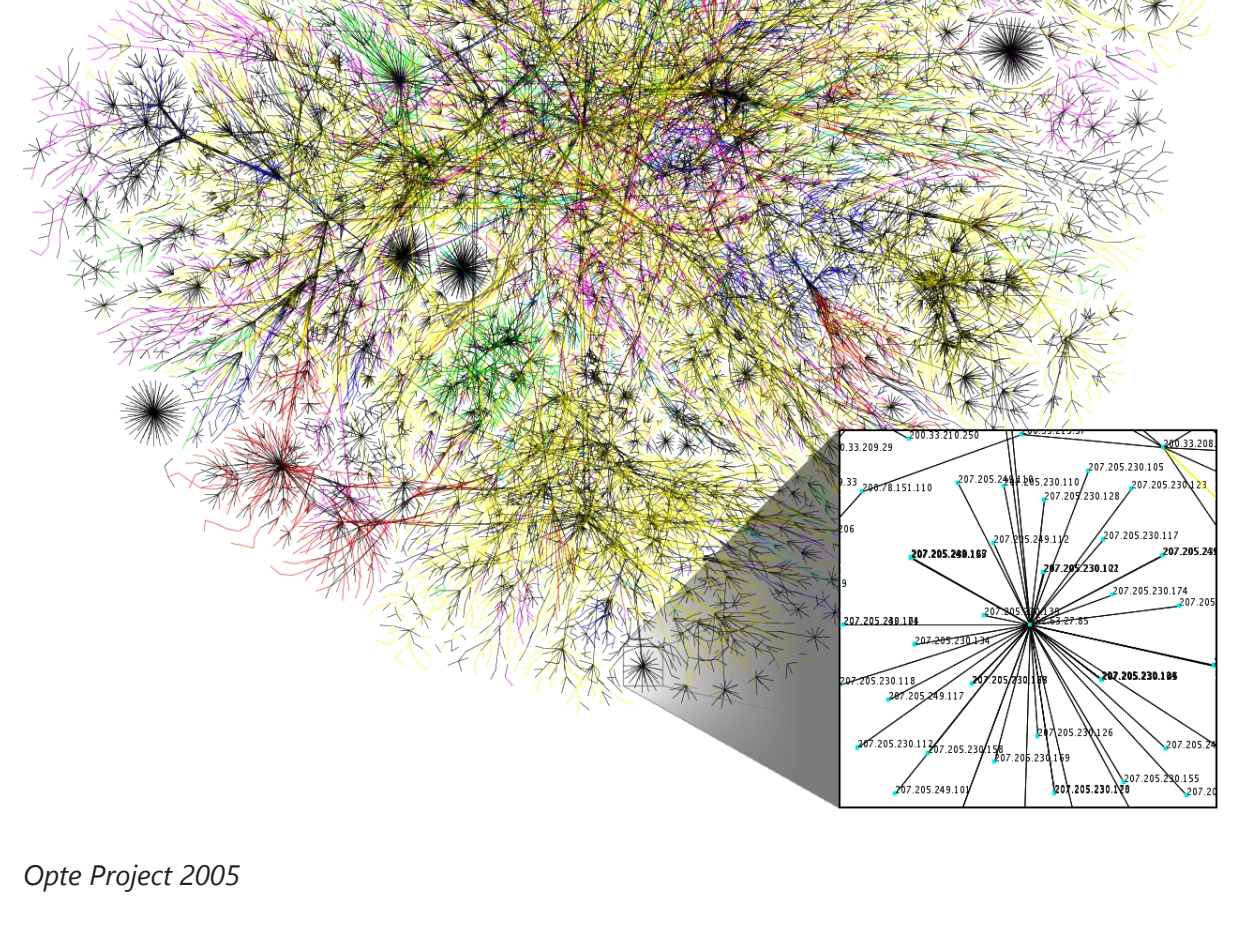
In programming tools like R or Python, it is easy to write code that repeats specicif commands. This allows one to send a huge amount of requests in a very short time.

This can be considered an attack on the server and is illegal!

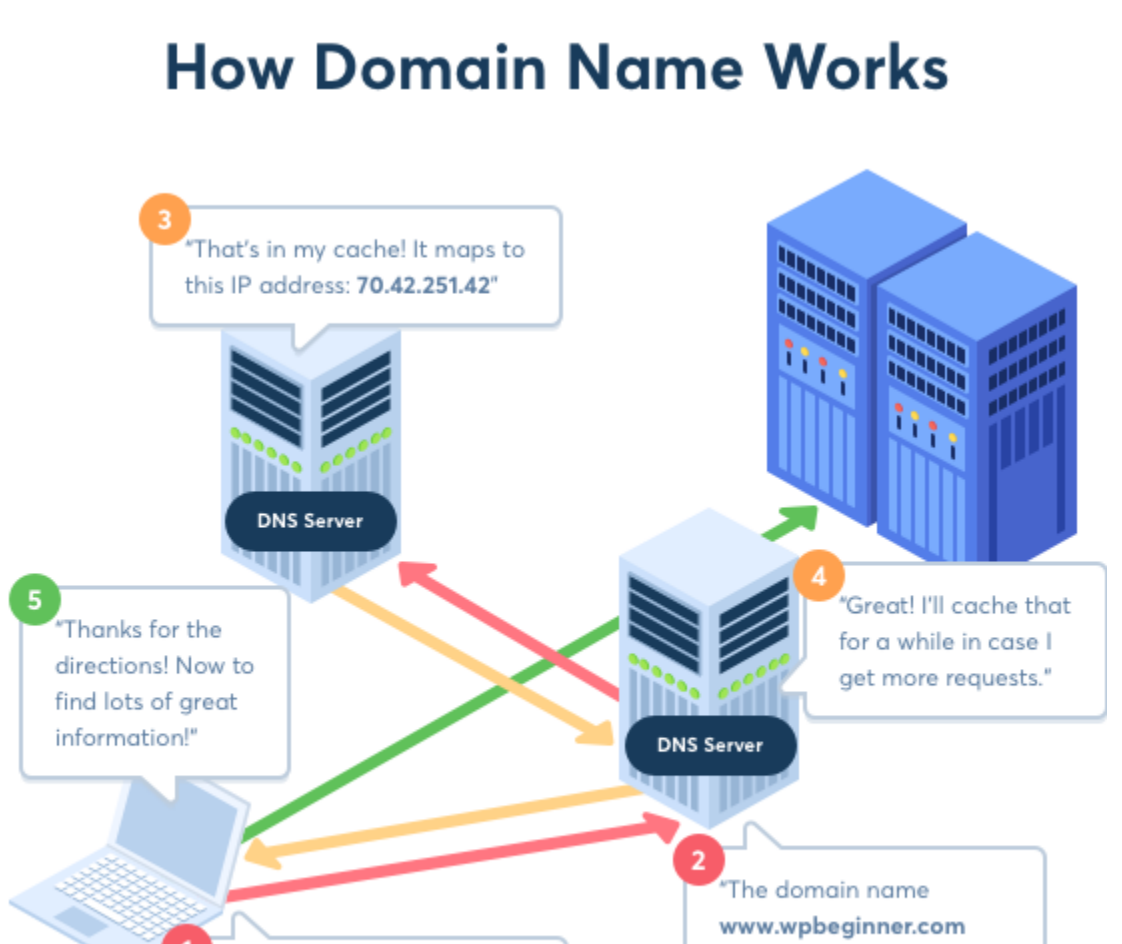
How the internet and browsers work (lightning fast edition)

The internet 101

- The internet: A global system of connected computers and servers ("Internet" short for "interconnected network")
- "(World Wide Web)" (WWW): A collection of of resources accessible via the internet
  - The "web" can be considered the navigatable and public parts of the internet
- Every computer or server on the internet has a unique ID in the form of a IP-address (fx 127.28.115.253)
  - IP: Internet Protocol
- When accessing the internet via the web, other computers and servers are contacted to retrieve information
- The web uses "URL's" to give unique and human-readable addresses to servers (instead of IP-addresses)
  - URL: Unuform Resource Locators
- The connection between an IP-address and a URL is handles by DNS servers
  - DNS: Domain Name System



Opte Project 2005



wpbeginner.com

What does this mean?

The way the internet is constructed has the following consequences:

- There is not a common "address book" for the internet (DNS servers are hosted by several providers)
- Accessing resources on the web means accessing some raw information on a server or computer

How does a browser work?

Accessing resources on the web means accessing some raw information. When you are asking for a website via a browser, you are therefore asking for the raw source code located on that server.

A "browser" is a program used for accessing the web, handling the requests to a specific URL, resolving DNS (which IP to which URL) and translating the raw source code to something human-readable.

When accessing the web in programming tools like R or Python, one has to replicate a lot of this functionality - especially the one relating to processing the raw source code.

Accessing the internet through R

`httr` can be used to access webpages (or, more generally, send requests over the internet).

In [ ]:
```
library(httr)

request <- GET("https://www.imdb.com/title/tt9820556/?ref_=tt_mv_close")
```

Did we reach the page?

In [3]:
```
status_code(request)
```

200

Yes!

Quick note on status codes

- Status codes beginning with 2 or 3: The request is successful
- Status codes beginning with 4: The request has failed (client-side, fx 404 when specifying a URL that does not exist on a given domain).
- Status codes beginning with 5: The request has failed (server-side)

Status codes can be used in code to check whether or not a site is reached before scraping.

Content of a webpage

The text from the webpage can be extracted as follows:

In [4]:
```
web_content = content(request, as = "text")
```

In [9]:
```
substr(web_content, 1, 2000) # first 2000 characters
```

```
'\n\n\n\n\n\n<!DOCTYPE html>\n<html\n    xmlns:og="http://ogp.me/ns#"\n
xmlns:fb="http://www.facebook.com/2008/fbml">\n    <head>\n        \n<script
type=\'text/javascript\'>var ue_t0=ue_t0||+new Date();</script>\n<script
type=\'text/javascript\'>\nwindow.ue_ihb = (window.ue_ihb || window.ueinit || 0) + 1;\nif (window.ue_ihb
=== 1) {\n\nvar ue_csm = window,\n    ue_hob = +new Date();\n(function(d){var e=d.ue=d.ue||
{},f=Date.now||function(){return+new Date};e.d=function(b){return f()-(b?0:d.ue_t0)};e.stub=function(b,a)
{if(!b[a]){var c=[];b[a]=function(){c.push([c.slice.call(arguments),e.d(),d.ue_id])};b[a].replay=function(b)
{for(var a;a=c.shift();)b(a[0],a[1],a[2])};b[a].isStub=1}};e.exec=function(b,a){return function(){try{return
b.apply(this,arguments)}catch(c){ueLogError(c,{attribution:a||"undefined",logLevel:"WARN"})}}}})
(ue_csm);\n\n\n    var ue_err_chan = \'jserr\';\n(function(d,e){function h(f,b){if(!(a.ec>a.mxe)&&f)
{a.ter.push(f);b=b||{};var
c=f.logLevel||b.logLevel;c&&c!==k&&c!==m&&c!==n&&c!==p||a.ec++;c&&c!=k||a.ecf++;b.pageURL=""+
(e.location?
e.location.href:"");b.logLevel=c;b.attribution=f.attribution||b.attribution;a.erl.push({ex:f,info:b})}}function
l(a,b,c,e,g){d.ueLogError({m:a,f:b,l:c,c:""+e,err:g,fromOnError:1,args:arguments},g?
{attribution:g.attribution,logLevel:g.logLevel}:void 0);return!1}var
k="FATAL",m="ERROR",n="WARN",p="DOWNGRADED",a={ec:0,ecf:0,\npec:0,ts:0,erl:[],ter:
[],mxe:50,startTimer:function(){a.ts++;setInterval(function()
{d.ue&&a.pec<a.ec&&d.uex("at");a.pec=a.ec},1E4)}};l.skipTrace=1;h.skipTrace=1;h.isStub=1;d.ueLogError=h;d.u
(ue_csm,window);\n\n\nvar ue_id = \'DSBSGY751A676ZSADFVQ\',\n    ue_url,\n    ue_navtiming = 1,\n
ue_mid = \'A1EVAM02EL8SFB\',\n    ue_sid = \'133-9166980-9444756\',\n    ue_sn = \'www.imdb.com\',\n
ue_furl = \'fls-na.amazon.com\',\n    ue_surl = \'https://unagi-
na.amazon.com/1/events/com.amazon.csm.nexusclient.prod\',\n    ue_int = 0,\n    ue_fcsn = 1,\n    ue_urt
= 3,\n    ue_rpl_ns = \'cel-rpl\',\n    ue_ddq = 1,\n    ue_fpf = \'//fls-na.amaz'
```

One could simply process the raw HTML with the use of something like regular expression but seeing as HTML has a structure, one can use that structure to extract specific information.

EXERCISE 1: Access the web in R

- Use the `GET()` function from `httr` to send a request to "https://www.imdb.com" and store the request as an object.
- Check the status code: Was the request successful?

A quick introduction to HTML

Instead of processing the HTML as raw text, we can utilize the structure of HTML to extract specific parts of a webpage.

This requires some knowledge of what HTML is and how it is structured.

HTML is short for "Hyper-Text Markup Language". It is used on webpages to give the page its structure.

HTML is structured in "tags" denoted by `<>` and `</>`. The tags denote what kind of content it is. `<p>` is for example a paragraph tag. A piece of HTML like: `<p> This is a paragraph </p>` will render the sentence "This is a paragraph" as a paragraph. Common tags include `h1` for headings (and `h2`, `h3` and so on), `a` for links and `div` for a division or section.

HTML is structured in a tree-like structure. Tags are therefore usually located within other tags. Tags on the same level is refered to as "siblings", tags inside other tags refered to as "children" and tags outside other tags refered to as "parents".

HTML uses "attributes" to both differentiate between the same type of tags and to add other variables/information to the tag. The `id` attribute is fx used to give several tags a common id. `class` is used to differentiate between different tags and provide them with different stylings. A common and useful attribute is `href` which contain the link that a hyperlink is refering to.

In [10]:
```
html <- deparse(
    '<html>
    <body>
        <div id="convo1">
            <p class="kenobi">Hello There!</p>
        </div>
        <div id="convo2">
            <p class="grievous">General Kenobi!</p>
        </div>
        <div id="convo3">
            <p class="kenobi">So Uncivilized!</p>
        </div>
    </body>
    </html>
    '
)
```

Inspecting HTML in your browser

Let's look at an upcoming movie ("Breach"): https://www.imdb.com/title/tt9820556/?ref_=tt_mv_close

A browser can be used to inspect the HTML used on the website.

EXERCISE 2: Identify HTML

Go to the "Coming Soon" section on IMDb: https://www.imdb.com/movies-coming-soon/?ref_=nv_mv_cs

- Inspect the HTML
- Determine the tag used for the titles of the upcoming films. Are they within other tags? What attributes can help navigate to the titles?

Simple HTML extraction using `rvest`

`rvest` contains various functions for extracting information from webpages using the HTML structure.

Let's look at the movie "Breach" again: https://www.imdb.com/title/tt9820556/?ref_=tt_mv_close

We start by loading it as an HTML/XML node object in R. This allows us to "navigate" the HTML inside R.

This can be done with `GET()` and `content()` but the function `read_html()` combines the two functions:

In [11]:
```
library(rvest)

breach_html <- read_html("https://www.imdb.com/title/tt9820556/?ref_=tt_mv_close")
```

```
Loading required package: xml2
Registered S3 method overwritten by 'rvest':
  method            from
  read_xml.response xml2
```

`html_nodes()` is the "core function" in `rvest`. This is used to extract specific parts of the HTML.

The title of the movie is located in a `h1` tag:

In [39]:
```
title_html <- html_nodes(breach_html, "h1")
```

This receives the HTML of that tag (including children)

In [40]:
```
title_html
```

```
{xml_nodeset (1)}
[1] <h1 class="">Anti-Life <span id="titleYear">(<a href="/year/2020/?ref_=tt ...
```

We can the retrieve the textual content:

In [44]:
```
title_text <- html_text(title_html, trim = TRUE) # Option to leading and trailing whi
title_text
```

'Anti-Life (2020)'

We can also inspect the attributes inside the tag with `html_attrs()`:

In [48]:
```
html_attrs(title_html)
```

1. **class:** ''

The `h1` tag itself only has a class attribute (with no value). However, there is an `a` child tag with an `href` attribute.

This can be extracted as follows:

In [68]:
```
link <- html_nodes(title_html, "a")
html_attr(link, "href")
```

'/year/2020/?ref_=tt_ov_inf'

EXERCISE 3: Extract HTML (specific node/tag)

Return to the "Coming Soon" section on IMDb: https://www.imdb.com/movies-coming-soon/?ref_=nv_mv_cs

- Using `html_nodes`, extract the titles of the upcoming movies.

Solution

In [62]:
```
upcoming <- read_html("https://www.imdb.com/movies-coming-soon/?ref_=nv_mv_cs")
titles_html <- html_nodes(upcoming, "h4")
upcoming_titles <- html_text(titles_html, trim = TRUE) # Returned as a list

upcoming_titles
```

1. 'November 20'
2. 'Jiu Jitsu (2020)'
3. 'The Last Vermeer (2019)'
4. 'Vanguard (2020)'
5. 'The Furies (2019)'
6. 'November 27'
7. 'The Croods: A New Age (2020)'
8. 'Zappa (2020)'
9. 'Dutch'

Extracting links to the movies

In [71]:
```
library(dplyr)
titles_html %>%
    html_nodes("a") %>%
    html_attr("href")
```

1. NA
2. '/title/tt9624766/?ref_=cs_ov_tt'
3. '/title/tt8337320/?ref_=cs_ov_tt'
4. '/title/tt9695722/?ref_=cs_ov_tt'
5. '/title/tt8228172/?ref_=cs_ov_tt'
6. NA
7. '/title/tt2850386/?ref_=cs_ov_tt'
8. '/title/tt4881578/?ref_=cs_ov_tt'
9. '/title/tt5738280/?ref_=cs_ov_tt'

Complex extraction using `rvest` and CSS selectors

So far we have extracted parts of HTML using specific tags. What we are acutally specifying is a "CSS selector". CSS selectors are ways of specifying HTML paths to extract specific parts of HTML.

This allows us to utilize the HTML tree structure and use the different attributes.

CSS selectors are specified as paths usually separated by spaces.

Fx on a movie page like the one for "Breach", the title has a link in the release year; i.e. there is an `a` tag inside the `h1` tag. This path can be specified with CSS like: `h1 a`. This selector finds all `a` tags that are children of an `h1` tag.

Here is some useful CSS selector notation:

For specifying specific tags:

- `.` for a class: `div.credit_summary_item` (selects `div` tags with the class `credit_summary_item`
- `#` for id: `div#title_completed_pro_link` (selects `div` tags with the id `title_completed_pro_link`
- `[attribute]` for an attribute: `[href]` (selects all tags with a `href` attribute.
- `[attribute = value]` for an attribute with a given value: `[name=slot_center-20]` selects all tags with a `name` attribute containing the vlaue `slot_center-20`.
  - Starts with: `^=`
  - Ends with: `$=`
  - Wildcards: `*=`
- `contains(text)` : Match based on text inside tag

These can be combined. The CSS selector: `div.plot_summary div.summary_text` selects the `div` tag with the plot summary on a movie page on IMDb.

For navigating the structure:

- `+` : Next sibling
- `~` : All siblings following
- `first-child` / `last-child` : Finds tags that are the first or last children respectively inside another tag
- `first-of-type` / `last-of-type` : Finds the first/last tag of a certain type
- `only-child` / `only-type` : Finds "lone children" or unique types
- `nth-child` / `nth-type()` : Specifying child or type based on their order (can be specified as a formula)

How to use CSS selectors

Looking at the movie page for "Breach", how can we extract the name of the director?

- The director's name is located in an `a` tag
- The `a` tag is the next siblibng of a `h4` tag with the class "inline"
- The `h4` tag contains the text "Director:"
- The `h4` and `a` tag are located inside a `div` tag with the class "credit_summary_item"

This can be written as the following CSS selector: `div.credit_summary_item h4.inline:contains(Director) + a`

In [73]:
```
director <- html_nodes(breach_html, "div.credit_summary_item h4.inline:contains(Direct
```

In [75]:
```
html_text(director)
```

'John Suits'

EXERCISE 4: 10 minute CSS training

Writing CSS selectors is a skill in of itself. Let's exercise that a bit.

The game "CSS Diner" is excellent for understanding the basics. See how far you can get in 10 minutes!

- CSS Diner: https://flukeout.github.io/

EXERCISE 5 (together): Extracting specific HTML content I

How can we extract the cast member currently on top of the cast list for "Breach"?

EXERCISE 6: Extracting specific HTML content II

Return to the "Coming Soon" section on IMDb: https://www.imdb.com/movies-coming-soon/?ref_=nv_mv_cs

Figure out a way to extract the following:

- The titles of the upcoming movies
- The directors of the upcoming movies
- The stars of the upcoming movies

Solution

In [83]:
```
movie_nodes <- html_nodes(upcoming, "td.overview-top")

titles <- movie_nodes %>%
    html_nodes("h4 a") %>%
    html_text()

directors <- movie_nodes %>%
    html_nodes("div.txt-block h5.inline:contains(Director) + span") %>%
    html_text()

stars <- list()

for (i in 1:length(movie_nodes)){
    movie_stars <- movie_nodes[i] %>%
    html_nodes("div.txt-block h5.inline:contains(Stars) ~ a") %>%
    html_text()
    stars[[i]] <- movie_stars
}
```

Converting to data frame

In [96]:
```
upcoming_df <- data.frame(titles, directors)
upcoming_df$stars <- stars
```

In [97]:
```
upcoming_df
```

titles	directors	stars
Jiu Jitsu (2020)	Dimitri Logothetis	Nicolas Cage , Marie Avgeropoulos, Frank Grillo , Tony Jaa
The Last Vermeer (2019)	Dan Friedkin	Daan Aufenacker , Claes Bang , Matt Beauman-Jones, Mark Behan
Vanguard (2020)	Stanley Tong	Jackie Chan, Yang Yang , Miya Muqi , Lun Ai
The Furies (2019)	Tony D'Aquino	Airlie Dodds , Linda Ngo , Taylor Ferguson, Ebony Vagulans
The Croods: A New Age (2020)	Joel Crawford	Emma Stone , Nicolas Cage , Ryan Reynolds, Leslie Mann
Zappa (2020)	Alex Winter	Frank Zappa , Steve Vai , Pamela Des Barres, Gail Zappa
Dutch	Preston A. Whitmore II	Natasha Marc , Jeremy Meeks , Robert Costanzo , Malcolm David Kelley

Further reading

- Other tools:
  - `Rcrawler` : https://github.com/salimk/Rcrawler