Swiftea-Crawler Documentation

Release 1.0.2

Thykof Huluti

Apr 08, 2018

Contents

Swiftea-Crawler is an open source web crawler for Swiftea. It can't be run by contributors because it needs private_data.py wich is not upload for obvious reason.

CHAPTER 1

Modules

Here are described the two executables of Swiftea-Crawler.

1.1 crawler main module

1.2 statistiques module

Display stats.

`stats.`**`stats`**(*dir_stats='data/stats/'*)

`stats.`**`compress_stats`**(*filename*)

`stats.`**`average`**(*content*)

Calculate average.

Parameters **`content`** (*`list`*) – values

Returns average

CHAPTER 2

Package

These packages provide all functions and class that crawler need.

2.1 swiftea_bot.module module

Define several functions for all crawler's class.

swiftea_bot.module.**tell**(*message*, *error_code=''*, *severity=1*)
Manage newspaper.

Print in console that program doing and save a copy with time in event file.

Parameters

- **message** (*str*) – message to print and write
- **error_code** (*int*) – (optional) error code, if given call errors() with given message
- **severity** (*int*) – 1 is default severity, -1 add 4 spaces befor message, 0 add 2 spaces befor the message, 2 uppercase and underline message.

swiftea_bot.module.**errors**(*message*, *error_code*)
Write the error report with the time in errors file.

Normaly call by tell() when a error_code parameter is given.

Parameters

- **message** (*str*) – message to print and write
- **error_code** (*int*) – error code

swiftea_bot.module.**create_dirs**()
Manage crawler's runing.

Test lot of things: create config directory

create doc file if doesn't exists

create config file if it doesn't exists

create links directory if it doesn't exists

create index directory if it doesn't exists

swiftea_bot.module.**def_links**()
Create directory of links if it doesn't exist

Ask to user what doing if there isn't basic links. Create a basic links file if user what it.

swiftea_bot.module.**is_index**()
Check if there is a saved inverted-index file.

Returns True if there is one

swiftea_bot.module.**can_add_doc**(*docs*, *new_doc*)
to avoid documents duplicate, look for all url doc.

Parse self.infos of Crawler and return True if new_doc isn't in it.

Parameters

- **docs** (*list*) – the documents to check
- **new_doc** (*dict*) – the doc to add

Returns True if can add the doc

swiftea_bot.module.**remove_duplicates**(*old_list*)
Remove duplicates from a list.

Parameters **old_list** (*list*) – list to clean

Returns list without duplicates

swiftea_bot.module.**stats_webpages**(*begining*, *end*)
Write the time in second to crawl 10 webpages.

Parameters

- **begining** (*int*) – time before starting crawl 10 webpages
- **end** (*int*) – time after crawled 10 webpages

swiftea_bot.module.**stats_send_index**(*begining*, *end*)
Time spent between two sending of index

swiftea_bot.module.**convert_keys**(*inverted_index*)
Convert *str* words keys into *int* from inverted-index.

Json convert doc id key in str, must convert in int.

Parameters **inverted_index** – inverted_index to convert

Tyep inverted_index dict

Returns converted inverted-index

2.2 swiftea_bot.data module

Define required data for crawler.

2.3 swiftea_bot.file_manager module

Swiftea-Crawler use lot a files. For example to manage configurations, stuck links... Here is a class who manager files of crawler.

class swiftea_bot.file_manager.**FileManager**

File manager for Swiftea-Crawler.

Save and read links, read and write configuration variables, read inverted-index from json file saved and from file using when send it.

Create configuration file if doesn't exists or read it.

check_stop_crawling()

Check if the user want to stop program.

save_config()

Save all configurations in config file.

save_links(*links*)

Save found links in file.

Save link in a file without doublons.

Parameters **links** (*list*) – links to save

ckeck_size_links(*links*)

Check number of links in file.

Parameters **links** (*str*) – links saved in file

check_size_files()

get_url()

Get url of next webpage.

Check the size of curent reading links and increment it if over.

Returns url of webpage to crawl

save_inverted_index(*inverted_index*)

Save inverted-index in local.

Save it in a .json file when can't send.

Parameters **inverted_index** (*dict*) – inverted-index

get_inverted_index()

Get inverted-index in local.

Call after a connxion error. Read a .json file conatin inverted-index. Delete this file after reading.

Returns inverted-index

read_inverted_index()

Get inverted-index in local.

Call after sending inverted-index without error. Read all files created for sending inverted-index.

Returns inverted-index

get_lists_words()

Get lists words from data

Check for dirs lists words, create it if don't exists.

Returns stopwords, badwords

2.4 crawling.web_connection module

Connection to webpage are manage with requests module. Thoses errors are waiting for: timeout with socket module and with urllib3 mudule and all RequestException errors.

class crawling.web_connection.**WebConnection**
Manage the web connection with the page to crawl.

get_code(*url*)
Get source code of given url.

Parameters **url** (*str*) – url of webpage

Returns source code, True if no take links, score and new url (redirection)

send_request(*url*)

search_encoding(*headers*, *code*)
Searche encoding of webpage in source code.

If an encoding is found in source code, score is 1, but if not score is 0 and encoding is utf-8.

Parameters

- **headers** (*dict*) – hearders of requests
- **code** (*str*) – source code

Returns encoding of webpage and it score

check_robots_perm(*url*)
Check robots.txt for permission.

Parameters **url** (*str*) – webpage url

Returns True if can crawl

duplicate_content(*request1*, *url*)
Avoid param duplicate.

Compare source codes with params and whitout. Return url whitout params if it's the same content.

Parameters **request** (*requests.models.Response*) – request

Returns url, source code

2.5 crawling.connection module

Define several functions WebConnection.

crawling.connection.**no_connection**(*url='https://github.com'*)
Check connection.

Try to connect to swiftea website.

Parameters **url** – url use by test

Returns True if no connection

crawling.connection.**is_nofollow**(*url*)
Check if take links.

Search !nofollow! at the end of url, remove it if found.

Parameters **url** (*str*) – webpage url

Returns True if nofollow and url

crawling.connection.**duplicate_content**(*code1*, *code2*)
Compare code1 and code2.

Parameters

- **code1** (*str*) – first code to compare
- **code2** (*str*) – second code to compare

crawling.connection.**all_urls**(*request*)
Return all urls from request.history.

Parameters

- **request** (*requests.models.Response*) – request
- **first** (*str*) – list start with the url if given

Returns list of redirected urls, first is the last one

2.6 crawling.site_informations module

After parse source code, data extracted must be classify and clean. Here is a class who use the html parser and manage all results.

class crawling.site_informations.**SiteInformations**
Class to manage searches in source codes.

set_listswords(*stopwords*, *badwords*)

get_infos(*url*, *code*, *nofollow*, *score*)
Manager all searches of webpage's informations.

Parameters

- **url** (*str*) – url of webpage
- **score** (*int*) – score of webpage
- **code** (*str*) – source code of webpage
- **nofollow** (*bool*) – if we take links of webpage

Returns links, title, description, key words, language, score, number of words

detect_language(*keywords*)
Detect language of webpage if not given.

Parameters **keywords** (*list*) – keywords of webpage used for detecting

Returns language found

clean_links(*links*, *base_url=None*)
Clean webpage's links: rebuild urls with base url and remove anchors, mailto, javascript, .index.

Parameters **links** (*list*) – links to clean

Returns cleanen links without duplicate

clean_favicon(*favicon*, *base_url*)
Clean favicon.

Parameters **favicon** (*str*) – favicon url to clean

Returns cleaned favicon

clean_keywords(*dirty_keywords*, *language*)
Clean found keywords.

Delete stopwords, bad chars, two letter less word and split word1-word2

Parameters **keywords** (*list*) – keywords to clean

Returns list of cleaned keywords

sane_search(*keywords*, *language*, *max_ratio=0.2*)
Filter adults websites.

Param keywords: webpage's keywords

Pram language found website language

Returns True or False

2.7 crawling.searches module

Define several functions SiteInformations.

crawling.searches.**clean_text**(*text*)
Clean up text by removing tabulation, blank and carriage return.

Parameters **text** (*str*) – text to clean_text

Returns cleaned text

crawling.searches.**get_base_url**(*url*)
Get base url using urlparse.

Parameters **url** (*str*) – url

Returns base url of given url

crawling.searches.**is_homepage**(*url*)
Check if url is the homepage.

If there is only two '/' and two '.' if www and one otherwise.

Parameters **url** (*str*) – url to check

Returns True or False

crawling.searches.**clean_link**(*url*, *base_url=None*)
Clean a link.

Rebuild url with base url, pass mailto and javascript, remove anchors, pass if more than 5 query, pass if more than 255 chars, remove /index.xxx, remove last /.

Parameters

- **url** (*str*) – links to clean
- **base_url** – base url for rebuilding, can be None if

Returns cleaned link

crawling.searches.**capitalize**(*text*)
Upper the first letter of given text

Parameters **text** (*str*) – text

Returns text

crawling.searches.**stats_links**(*stats*)
Write the number of links for statistics.

Parameters **stat** (*int*) – number of list in a webpage

2.8 crawling.parsers module

Data of webpage are geted by the python html.parser. Here is two parser, the first one for all informations and the sencond one only for encoding.

class crawling.parsers.**ExtractData**
Bases: html.parser.HTMLParser

Html parser for extract data.

self.object : the type of text for title, description and keywords

dict(attrs).get('content') : convert attrs in a dict and retrun the value

Data could be extract: title

language

description

links with nofollow and noindex

stylesheet

favicon

keywords: h1, h2, h3, strong, em

re_init()
Call when met html tag, put back all variables to default.

handle_starttag(*tag*, *attrs*)
Call when parser met a starting tag.

Parameters

- **tag** (*str*) – starting tag
- **attrs** (*list*) – attributes: [('name', 'language'), ('content', 'fr')]

handle_data(*data*)
Call when parser met data.

Parameters **tag** (*str*) – starting tag

handle_endtag(*tag*)
Call when parser met a ending tag.

Parameters

- **tag** (*str*) – starting tag

- **attrs** (*list*) – attributes

handle_entityref(*name*)

handle_charref(*name*)

crawling.parsers.**meta**(*attrs*)
Manager searches in meat tag.

Can find: <meta name='description' content='my description'/>

<meta name='language' content='en'/>

<meta http-equiv='content-language' content='en'/>

Apram attrs attributes of meta tag

Returns language, description, object

crawling.parsers.**can_append**(*url*, *rel*)
Check rel attrs to know if crawler can take this the link.

Add !nofollow! at the end of the url if can't follow links of url.

Parameters

- **url** (*str*) – url to add
- **rel** (*str*) – rel attrs in a tag

Returns None if can't add it, otherwise return url

class crawling.parsers.**ExtractEncoding**
Bases: html.parser.HTMLParser

Html parser for extract encoding from source code.

handle_starttag(*tag*, *attrs*)
Call when parser met a starting tag.

Parameters

- **tag** (*str*) – starting tag
- **attrs** (*list*) – attributes

2.9 database.database module

Define several functions for DatabaseSwiftea.

database.database.**url_is_secure**(*url*)
Check if given url is secure (https).

Parameters **url** (*str*) – url to check

Returns True if url is secure

database.database.**convert_secure**(*url*)
Convert https to http and http to https.

Parameters **url** (*str*) – url to convert

Returns converted url

2.10 database.database_manager module

class database.database_manager.**DatabaseManager**(*host*, *user*, *password*, *name*)
Class to manage query to Database using PyMySQL.

How to: create a subclass

result, response = self.send_comand(command, data=tuple(), all=False)

if 'error' in response:

print('An error occured.')

where result are data asked and response a message.

Parameters

- **host** (*str*) – hostname of the sftp server
- **user** (*str*) – username to use for connection
- **password** (*str*) – password to use for connection
- **name** (*str*) – name of database

set_name(*name*)
Set base name

Parameters **name** (*str*) – new base name

connection()
Connect to database.

close_connection()
Close database connection.

send_command(*command*, *data=()*, *fetchall=False*)
Send a query to database.

Catch timeout and OperationalError.

Parameters

- **data** (*tuple*) – data attached to query
- **fetchall** (*bool*) – True if return all results

Returns result of the query and status message

2.11 database.database_swiftea module

class database.database_swiftea.**DatabaseSwiftea**(*host*, *user*, *password*, *name*, *table*)
Bases: *database.database_manager.DatabaseManager*

Class to manage Swiftea database.

Parameters

- **host** (*str*) – hostname of the sftp server
- **user** (*str*) – username to use for connection
- **password** (*str*) – password to use for connection
- **name** (*str*) – name of database

send_doc(*webpage_infos*)
send documents informations to database.

Parameters **infos** (*list*) – informations to send to database

Returns True if an error occured

update(*infos*, *popularity*)
Update a document in database.

Parameters

- **infos** (*dict()*) – doc infos
- **popularity** (*int*) – new doc popularity

Returns True is an arror occured

insert(*infos*)
Insert a new document in database.

Parameters **infos** (*dict()*) – doc infos

Returns True is an arror occured

get_doc_id(*url*)
Get id of a document in database.

Parameters **url** (*str*) – url of webpage

Returns id of webpage or None if not found

del_one_doc(*url*, *table=None*)
Delete document corresponding to url.

Parameters **url** (*str*) – url of webpage

Returns status message

suggestions()
Get the five first URLs from Suggestion table and delete them.

Returns list of url in Suggestion table and delete them

doc_exists(*url*)
Check if *url* is in database.

Parameters **url** (*str*) – url corresponding to doc

Returns True if doc exists

https_duplicate(*old_url*)
Avoid https and http duplicate.

If old url is secure (https), must delete insecure url if exists, then return secure url (old url). If old url is insecure (http), must delete it if secure url exists, then return secure url (new url)

Parameters **old_url** (*str*) – old url

Returns url to add and url to delete

2.12 index.index module

Define several functions for inverted-index.

index.index.**count_files_index**(*index*)
Return number of file to download are uplaod

Parse languages and letters from the given index.

Returns int

index.index.**stats_dl_index**(*begining*, *end*)
Write the time to download inverted-index.

Parameters

- **begining** (*int*) – time download inverted-index
- **end** (*int*) – time after download inverted-index

index.index.**stats_ul_index**(*begining*, *end*)
Write the time to upload inverted-index.

Parameters

- **begining** (*int*) – time before send inverted-index
- **end** (*int*) – time after send inverted-index

2.13 index.inverted_index module

class index.inverted_index.**InvertedIndex**
Manage inverted-index for crawler.

Inverted-index is a dict, each keys are language

-> values are a dict, each keys are first letter

-> values are dict, each keys are two first letters

-> values are dict, each keys are word

-> values are dict, each keys are id

-> values are int : tf

example: ['FR']['A']['av']['avion'][21] is tf of word 'avion' in doc 21 in french.

setInvertedIndex(*inverted_index*)
Define inverted-index at the beginning.

Parameters **inverted_index** (*dict*) – inverted-index

getInvertedIndex()

Returns inverted-index

add_doc(*keywords*, *doc_id*, *language*)
Add all words of a doc in inverted-index.

Parameters

- **keywords** (*list*) – all word in doc_id
- **doc_id** (*int*) – id of the doc in database
- **language** (*str*) – language of word

add_word(*word_infos*, *doc_id*, *nb_words*)
Add a word in inverted-index.

Parameters

- **word_infos** (*dict*) – word infos: word, language, occurence, first letter and two first letters
- **doc_id** (*int*) – id of the doc in database
- **nb_words** (*int*) – number of words in the doc_id

delete_word(*word*, *language*, *first_letter*, *filename*)
Delete a word in inverted-index.

Parameters

- **word** (*str*) – word to delete
- **language** (*str*) – language of word
- **first_letter** (*str*) – first letter of word
- **filename** (*str*) – two first letters of word

delete_id_word(*word_infos*, *doc_id*)
Delete a id of a word in inverted-index

This method delete a word from a document. Remove a words from a doc.

Parameters

- **word_infos** (*dict*) – word infos: word, language, first letter and two first letters
- **doc_id** (*int*) – id of the doc in database

delete_doc_id(*doc_id*)
Delete a id in inverted-index.

Parameters **doc_id** (*int*) – id to delete

2.14 index.sftp_manager module

2.15 index.sftp_swiftea module

CHAPTER 3

Tests

Tests for Swiftea-Crawler using pytest.

3.1 tests.run_tests

3.2 tests.swiftea_bot_test

3.3 tests.crawling_test

3.4 tests.database_test

3.5 tests.index_test

3.6 tests.crawler_test

3.7 tests.global_test

3.8 tests.test_data

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

d

i

s

Index

G

H

I

M

N

R

S

T

U

W