Hyper-Text Query Language Python Interface

2011-03-24

1. About

Hyper-Text Query Language (HTQL) is a language for the query and extraction HTML data.
This guide explains the use of HTQL Python interface for use in Python applications.

In addition, the HTQL Python package includes several text search libraries that are implemented in C++ and optimized for efficient search of large-volume data, including regular expression search, dictionary search, similar string search, and text clustering. This guide explains their usage.

Specifically, the HTQL Python interface includes:

1) An HTQL module to query and extract content in HTQL language;
2) A Browser module to crawl websites;
3) A RegEx module for regular expression;
4) A Dict module for dictionary support;
5) An Align module for string matching;
6) A Tool module with some helper functions.

HTQL specific syntax can be found at http://htql.net/htql-manual.pdf.

2. Installation

Download the htql.zip (v2.7) or htql.zip (v3.2) and extract the htql.pyd to Python's DLLs directory, such as in 'C:\Python27\DLLs\' or 'C:\Python32\DLLs\'.

3. Simple Examples

A simple example to extract url and text from links.

```
import htql;
page="<a href=a.html>1</a><a href=b.html>2</a><a href=c.html>3</a>";
query="<a>:href,tx";

for url, text in htql.HTQL(page, query):
    print(url, text);
```

An example using htql.Browser to Bing search and get the first link page:

```
import htql;
a=htql.Browser();
b=a.goUrl("http://www.bing.com/");
c=a.goForm("<form>1", {"q":"test"});
for d in htql.HTQL(c[0], "<a (tx like '%test%')>"):
   print(d);
e=a.click("<a (tx like '%test%' and not (href like '/search%'))>1");
```

If you have installed IRobotSoft Web Scraper, you can browse the web visually with:

```
a=htql.Browser(2);
```

An example to parse state and zip from US address using HTQL regular expression:

```
import htql;
address = '88-21 64th st , Rego Park , New York 11374'
states=['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut',
        'Delaware', 'District Of Columbia', 'Florida', 'Georgia', 'Hawaii',  'Idaho',
        'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland',
        'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana',
        'Nebraska', 'Nevada',  'New Hampshire', 'New Jersey', 'New Mexico', 'New York',
        'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma',  'Oregon', 'PALAU',
        'Pennsylvania', 'PUERTO RICO', 'Rhode Island', 'South Carolina', 'South Dakota',
        'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington', 'West Virginia',
        'Wisconsin', 'Wyoming'];

a=htql.RegEx();
a.setNameSet('states', states);

state_zip1=a.reSearchStr(address, "&[s:states][,\s]+\d{5}", case=False)[0];
# state_zip1 = 'New York 11374'

state_zip2=a.reSearchList(address.split(), r"&[ws:states]<,>?<\d{5}>", case=False)[0];
# state_zip2 = ['New', 'York', '11374']
```

4. HTQL Python Interface

4.1 HTQL interface

htql.HTQL(page, query) returns tuple iterator

Query the page with HTQL

htql.query(page, query) returns a list of tuples.

Query the page with HTQL

4.2 Browser interface

htql.Browser(type) returns htql.Browser

Create an HTQL browser. When type==1, returns a socket browser; when type==2, returns an IRobot browser

htql.Browser.goUrl(url, {name:value}, {cookie:value}, {http_header:value}) returns (page, url)

Go to URL page.

htql.Browser.goForm(form_htql, {name:value}, {cookie:value}, {http_header:value}) returns (page, url)

Submit a form.

htql.Browser.click(item_htql, wait) returns (page, url)

Click an item.

htql.Browser.getPage() returns (page, url)

Get the current page in browser.

htql.Browser.getUpdatedPage() returns (page, url)

Get the current page in browser after javascript has executed.

htql.Browser.runFunction(command) returns function result

Run a function command. Refer IRobot manual for available functions.

htql.Browser.setTimeout (connect, transfer)

Set browser timeout for connect and transfer, in seconds. Default 120 seconds for both.

htql.Browser.connectBrowser(host, port, [release=0])

Connect to an IRobot proxy browser. Host: the machine that runs the proxy; port: the proxy port; release: to quit the proxy (1) or not (0) after done, or don't care (-1).

4.3 Regular expression interface

htql.RegEx()

Create an HTQL Regular Expression object

htql.RegEx.setNameSet(name,[s], keep=0) returns total items in the name set.

Give a list of strings *s* a name *name*. The set can be referred from regular expression by '&[s:name]' or '&[ws:name]'

name: name of the string or regular expression set

s: a list of string, or regular expression strings

keep: keep the name in memory, do not reparse the set s if it is found in memory; this is to save parsing time.

htql.RegEx.reSearchStr(text, regex, case=True, overlap=False, useindex=False, keep=False, keepname=None) returns list of matching strings.

Do a regular expression search for text.

text: text to search

regex: regular expression

case: case sensitive (True) or not (False)

overlap: allow result strings to be overlapping (True) or not (False)

useindex: returns the (position, len) of the matching strings instead

keep: keep the regex parser in memory, do not reparse the regex if it is found in memory; this is to save the parsing time.

keepname: keep the regex parser in memory with this name.

htql.RegEx.reSearchList(text, regex, case=True, overlap=False, useindex=False, keep=False, keepname=None) returns list of matching sublist.

Do a regular expression search for sublist.

text: text tokens as a list of strings, e.g., ['aa', 'bb', 'cc','dd','ee']

regex: regular expression, with each token enclosed by '<' and '>'. E.g.: '<a.*>.*<c+>'

keep: keep the regex parser in memory, do not reparse the regex if it is found in memory; this is to save the parsing time.

keepname: keep the regex parser in memory with this name.

htql.RegEx.setExprStr(regex, case=True, keep=False, keepname=None) returns error code.

Set the regular expression to search

regex: regular expression for string search

case: case sensitive (True) or not (False)

keep: keep the regex parser in memory, do not reparse the regex if it is found in memory; this is to save the parsing time.

keepname: keep the regex parser in memory with this name.

htql.RegEx.setExprList(regex, case=True, keep=False, keepname=None) returns error code.

regex: regular expression for list search, with each token enclosed by '<' and '>'. E.g.: '<a.*>.*<c+>'

case: case sensitive (True) or not (False)

keep: keep the regex parser in memory, do not reparse the regex if it is found in memory; this is to save the parsing time.

keepname: keep the regex parser in memory with this name.

htql.RegEx.reExprSearch(text, overlap=False, useindex=False, exprname=None) returns error code.

text: text to search

overlap: allow result strings to be overlapping (True) or not (False)

useindex: returns the (position, len) of the matching strings instead

exprname: use the regex parser with this name if it is found in memory.

Regular expression: &[s:name]

To match from a set of strings in variable 'name'

Regular expression: &[ws:name]

To match from a set of string words in variable 'name'; each string word is split into tokens to match the list input.

Regular expression: &[s:name(string|string|...)]

To match from a set of strings and set them in variable 'name' at the same time

4.4 Dict interface

htql.Dict()

Create an HTQL Dict object

Htql.Dict.setDict(dictionary)

Set/add dictionary entries

Htql.Dict.searchKeys(text)

Search the occurring of dictionary words in *text*.

4.5 Align interface

htql.Align()

Create an HTQL Align object

htql.Align.align(str1, str2, case=True) returns *(cost, [str1_aligned, str2_aligned])*

Align two strings *str1* and *str2* based on their edit distance using dynamic algorithm.

str1: first string

str2: second string

case: to compare letters case sensitively or not

cost: The indel cost for the resulting alignment, where each *match*, *insert*, *delete*, and *replace* costs 0, 1, 1, and 2 respectively

str1_align returns the resulting alignments for *str1*, with filled-in '-' for gaps

str2_align returns the resulting alignments for *str2*, with filled-in '-' for gaps

htql.Align.minAlign(str1, list2, case=True, get_alignment=False) returns *(list_i, cost, [str1_aligned, str2_aligned])*

Align string *str1* to a list of strings in list2 to find the minimal cost alignment.

str1: String for alignment

list2: A list of strings for comparison

case: Compare letters in case sensitive way or not

get_alignment: whether to return the resulting alignment

list_i: The index of string *list2*, or *list2[list_i]*, that is most similar to *str1*

cost: The indel cost for the best alignment, where each *match*, *insert*, *delete*, and *replace* costs 0, 1, 1, and 2 respectively

str1_align returns the resulting alignments for *str1*, with '-' standing for gap

str2_align returns the resulting alignments for *str2*, with '-' standing for gap

htql.Align.cluster(text, is_html=False) returns *[(index1, index2)…]*

Cluster a list of input strings based on their similarity. Use getAlignment() function to get the resulting alignments

text: a list of string for clustering

is_html: whether the text strings are in HTML format

(index1,index2): the closest matches

htql.Align.getAlignment() returns *aligned_text*

Return the alignment by cluster() function.

aligned_text: the resulting alignment, with '-' standing for gap

4.6 Tools interface

htql.Tools()

Create an HTQL Tools object

htql.Tools.reSearchStr(text, regex, case, overlap, useindex) returns list of matching strings.

text: text to search

regex: regular expression

case: case sensitive (True) or not (False)

overlap: allow result strings to be overlapping (True) or not (False)

useindex: returns the (position, len) of the matching strings instead.

htql.Tools.reSearchList(text, regex, case, overlap, useindex) returns list of matching sublist.

text: text tokens as a list of strings, e.g., ['aa', 'bb', 'cc','dd','ee']

regex: regular expression, with each token enclosed by '<' and '>'. E.g.: '<a.*>.*<c+>'

5. Reference

HTQL was designed and created by Dr. Liangyou Chen as part of his Ph.D. dissertation work:

Ad Hoc Integration and Querying of Heterogeneous Online Distributed Databases. 2004. Mississippi State University.

Please cite the dissertation for references.