dryscrape Documentation

Release 0.8

Niklas Baumstark

December 27, 2016

Contents

dryscrape is a lightweight web scraping library for Python. It uses a headless Webkit instance to evaluate Javascript on the visited pages. This enables painless scraping of plain web pages as well as Javascript-heavy "Web 2.0" applications like Facebook.

It is built on the shoulders of capybara-webkit's webkit-server. A big thanks goes to thoughtbot, inc. for building this excellent piece of software!

CHAPTER 1

Contents

1.1 Installation

1.1.1 Prequirements

Before installing dryscrape, you need to install some software it depends on:

- Qt4, QtWebKit
- lxml
- pip

On Ubuntu you can do that with one command (the # indicates that you need root privileges for this):

```
# apt-get install libqt4-dev libqtwebkit-dev qt4-qmake build-essential \
                  python-lxml python-pip
```

On other operating systems, you can use pip to install lxml (though you might have to install libxml and the Python headers first).

1.1.2 dryscrape

First, get a copy of dryscrape using Git:

```
$ git clone https://github.com/niklasb/dryscrape.git dryscrape
$ cd dryscrape
```

To install dryscrape, you first need to install webkit-server. You can use pip to do this for you (while still in the dryscrape directory).

```
# pip install -r requirements.txt
```

If you want, you can of course also install the dependencies manually.

Afterwards, you can use the `setup.py` script included to install dryscrape:

```
# python setup.py install
```

Note that dryscrape is written for Python 2.7+, so make sure that you use both the `python` and `pip` commands in the right version.

1.2 Usage

1.2.1 First demonstration

A code sample tells more than thousand words:

```
import dryscrape

search_term = 'dryscrape'

# set up a web scraping session
sess = dryscrape.Session(base_url = 'http://google.com')

# we don't need images
sess.set_attribute('auto_load_images', False)

# visit homepage and search for a term
sess.visit('/')
q = sess.at_xpath('//*[@name="q"]')
q.set(search_term)
q.form().submit()

# extract all links
for link in sess.xpath('//a[@href]'):
  print link['href']

# save a screenshot of the web page
sess.render('google.png')
print "Screenshot written to 'google.png'"
```

In this sample, we use dryscrape to do a simple web search on Google. Note that we set up a Webkit driver instance here and pass it to a dryscrape *Session* in the constructor. The session instance then passes every method call it cannot resolve – such as `visit()`, in this case – to the underlying driver.

1.2.2 A more complex example

There was nothing much special about the example above. Let's look at a more advanced example that actually works on a Javascript-only application: GMail.

```
import time
import dryscrape

#========================================
# Setup
#========================================

email    = 'YOUR_USERNAME_HERE@gmail.com'
password = 'YOUR_PASSWORD_HERE'

# set up a web scraping session
sess = dryscrape.Session(base_url = 'https://mail.google.com/')

# there are some failing HTTP requests, so we need to enter
# a more error-resistant mode (like real browsers do)
sess.set_error_tolerant(True)
```

```
# we don't need images
sess.set_attribute('auto_load_images', False)

# if we wanted, we could also configure a proxy server to use,
# so we can for example use Fiddler to monitor the requests
# performed by this script
#sess.set_proxy('localhost', 8888)

#=========================================
# GMail send a mail to self
#=========================================

# visit homepage and log in
print "Logging in..."
sess.visit('/')

email_field    = sess.at_css('#Email')
password_field = sess.at_css('#Passwd')
email_field.set(email)
password_field.set(password)

email_field.form().submit()

# find the COMPOSE button and click it
print "Sending a mail..."
compose = sess.at_xpath('//*[contains(text(), "COMPOSE")]')
compose.click()

# compose the mail
to      = sess.at_xpath('//*[@name="to"]', timeout=10)
subject = sess.at_xpath('//*[@name="subject"]')
body    = sess.at_xpath('//*[@name="body"]')

to.set(email)
subject.set("Note to self")
body.set("Remember to try dryscrape!")

# send the mail

# seems like we need to wait a bit before clicking...
# Blame Google for this ;)
time.sleep(3)
send = sess.at_xpath('//*[normalize-space(text()) = "Send"]')
send.click()

# open the mail
print "Reading the mail..."
mail = sess.at_xpath('//*[normalize-space(text()) = "Note to self"]',
                     timeout=10)
mail.click()

# sleep a bit to leave the mail a chance to open.
# This is ugly, it would be better to find something
# on the resulting page that we can wait for
time.sleep(3)

# save a screenshot of the web page
print "Writing screenshot to 'gmail.png'"
```

```
sess.render('gmail.png')
```

This *just works*.

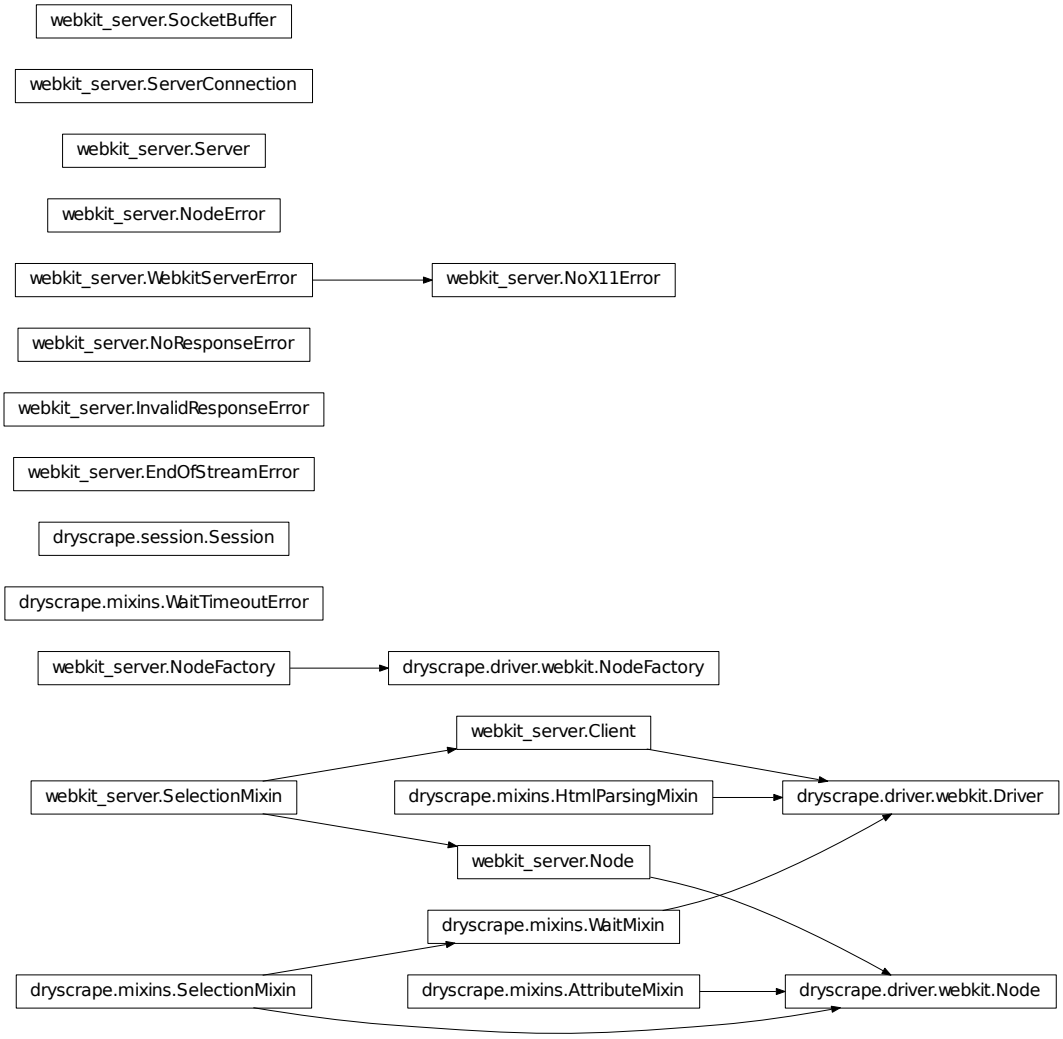
There are some things to note about it, though:

- `at_xpath()` and `at_css()` take an optional *timeout* argument that can be used to leave the application a bit of time to load content
- XPath is really useful, you should make yourself familiar with it. You can also use CSS, however.

1.3 API Documentation

This documentation also contains the API docs for the `webkit_server` module, for convenience (and because I am too lazy to set up dedicated docs for it).

1.3.1 Overview



1.3.2 Module `dryscrape.session`

class `dryscrape.session.`**`Session`**(*driver=None*, *base_url=None*)

Bases: `object`

A web scraping session based on a driver instance. Implements the proxy pattern to pass unresolved method calls to the underlying driver.

If no *driver* is specified, the instance will create an instance of `dryscrape.session.DefaultDriver` to get a driver instance (defaults to `dryscrape.driver.webkit.Driver`).

If *base_url* is present, relative URLs are completed with this URL base. If not, the *get_base_url* method is called on itself to get the base URL.

`complete_url`(*url*)

Completes a given URL with this instance's URL base.

interact(***local*)
Drops the user into an interactive Python session with the `sess` variable set to the current session instance. If keyword arguments are supplied, these names will also be available within the session.

visit(*url*)
Passes through the URL to the driver after completing it using the instance's URL base.

1.3.3 Module `dryscrape.mixins`

Mixins for use in dryscrape drivers.

class `dryscrape.mixins.`**`AttributeMixin`**
Bases: `object`

Mixin that adds `[]` access syntax sugar to an object that supports a `set_attr` and `get_attr` method.

class `dryscrape.mixins.`**`HtmlParsingMixin`**
Bases: `object`

Mixin that adds a `document` method to an object that supports a `body` method returning valid HTML.

document()
Parses the HTML returned by `body` and returns it as an lxml.html document. If the driver supports live DOM manipulation (like webkit_server does), changes performed on the returned document will not take effect.

class `dryscrape.mixins.`**`SelectionMixin`**
Bases: `object`

Mixin that adds different methods of node selection to an object that provides an `xpath` method returning a collection of matches.

at_css(*css*)
Returns the first node matching the given CSSv3 expression or `None`.

at_xpath(*xpath*)
Returns the first node matching the given XPath 2.0 expression or `None`.

children()
Returns the child nodes.

css(*css*)
Returns all nodes matching the given CSSv3 expression.

form()
Returns the form wherein this node is contained or `None`.

parent()
Returns the parent node.

class `dryscrape.mixins.`**`WaitMixin`**
Bases: *`dryscrape.mixins.SelectionMixin`*

Mixin that allows waiting for conditions or elements.

at_css(*css*, *timeout=1*, ***kw*)
Returns the first node matching the given CSSv3 expression or `None` if a timeout occurs.

at_xpath(*xpath*, *timeout=1*, ***kw*)
Returns the first node matching the given XPath 2.0 expression or `None` if a timeout occurs.

wait_for(*condition*, *interval=0.5*, *timeout=10*)
Wait until a condition holds by checking it in regular intervals. Raises `WaitTimeoutError` on timeout.

wait_for_safe(**args*, ***kw*)
Wait until a condition holds and return `None` on timeout.

wait_while(*condition*, **args*, ***kw*)
Wait while a condition holds.

exception `dryscrape.mixins.`**WaitTimeoutError**
Bases: `exceptions.Exception`

Raised when a wait times out

1.3.4 Module dryscrape.driver.webkit

Headless Webkit driver for dryscrape. Wraps the `webkit_server` module.

class `dryscrape.driver.webkit.`**Driver**(***kw*)
Bases: *webkit_server.Client*, *dryscrape.mixins.WaitMixin*, *dryscrape.mixins.HtmlParsingMixin*

Driver implementation wrapping a `webkit_server` driver.

Keyword arguments are passed through to the underlying `webkit_server.Client` constructor. By default, *node_factory_class* is set to use the dryscrape node implementation.

class `dryscrape.driver.webkit.`**Node**(*client*, *node_id*)
Bases: *webkit_server.Node*, *dryscrape.mixins.SelectionMixin*, *dryscrape.mixins.AttributeMixin*

Node implementation wrapping a `webkit_server` node.

class `dryscrape.driver.webkit.`**NodeFactory**(*client*)
Bases: *webkit_server.NodeFactory*

overrides the NodeFactory provided by `webkit_server`.

1.3.5 Module webkit_server

Python bindings for the webkit-server

class `webkit_server.`**Client**(*connection=None*, *node_factory_class=<class 'webkit_server.NodeFactory'>*)
Bases: *webkit_server.SelectionMixin*

Wrappers for the webkit_server commands.

If *connection* is not specified, a new instance of `ServerConnection` is created.

node_factory_class can be set to a value different from the default, in which case a new instance of the given class will be used to create nodes. The given class must accept a client instance through its constructor and support a `create` method that takes a node ID as an argument and returns a node object.

body()
Returns the current DOM as HTML.

clear_cookies()
Deletes all cookies.

clear_proxy()
Resets custom HTTP proxy (use none in future requests).

cookies()
Returns a list of all cookies in cookie string format.

eval_script(*expr*)
Evaluates a piece of Javascript in the context of the current page and returns its value.

exec_script(*script*)
Executes a piece of Javascript in the context of the current page.

get_node_factory()
Returns the associated node factory.

get_timeout()
Return timeout for every webkit-server command

headers()
Returns a list of the last HTTP response headers. Header keys are normalized to capitalized form, as in *User-Agent*.

issue_node_cmd(**args*)
Issues a node-specific command.

render(*path*, *width=1024*, *height=1024*)
Renders the current page to a PNG file (viewport size in pixels).

reset()
Resets the current web session.

reset_attribute(*attr*)
Resets a custom attribute.

set_attribute(*attr*, *value=True*)
Sets a custom attribute for our Webkit instance. Possible attributes are:

- `auto_load_images`
- `dns_prefetch_enabled`
- `plugins_enabled`
- `private_browsing_enabled`
- `javascript_can_open_windows`
- `javascript_can_access_clipboard`
- `offline_storage_database_enabled`
- `offline_web_application_cache_enabled`
- `local_storage_enabled`
- `local_storage_database_enabled`
- `local_content_can_access_remote_urls`
- `local_content_can_access_file_urls`
- `accelerated_compositing_enabled`
- `site_specific_quirks_enabled`

For all those options, `value` must be a boolean. You can find more information about these options in the QT docs.

set_cookie(*cookie*)
Sets a cookie for future requests (must be in correct cookie string format).

set_error_tolerant(*tolerant=True*)
DEPRECATED! This function is a no-op now.

Used to set or unset the error tolerance flag in the server. If this flag as set, dropped requests or erroneous responses would not lead to an error.

set_header(*key*, *value*)
Sets a HTTP header for future requests.

set_html(*html*, *url=None*)
Sets custom HTML in our Webkit session and allows to specify a fake URL. Scripts and CSS is dynamically fetched as if the HTML had been loaded from the given URL.

set_proxy(*host='localhost'*, *port=0*, *user=''*, *password=''*)
Sets a custom HTTP proxy to use for future requests.

set_timeout(*timeout*)
Set timeout for every webkit-server command

set_viewport_size(*width*, *height*)
Sets the viewport size.

source()
Returns the source of the page as it was originally served by the web server.

status_code()
Returns the numeric HTTP status of the last response.

url()
Returns the current location.

visit(*url*)
Goes to a given URL.

exception webkit_server.**EndOfStreamError**(*msg='Unexpected end of file'*)
Bases: exceptions.Exception

Raised when the Webkit server closed the connection unexpectedly.

exception webkit_server.**InvalidResponseError**
Bases: exceptions.Exception

Raised when the Webkit server signaled an error.

exception webkit_server.**NoResponseError**
Bases: exceptions.Exception

Raised when the Webkit server does not respond.

exception webkit_server.**NoX11Error**
Bases: *webkit_server.WebkitServerError*

Raised when the Webkit server cannot connect to X.

class webkit_server.**Node**(*client*, *node_id*)
Bases: *webkit_server.SelectionMixin*

Represents a DOM node in our Webkit session.

client is the associated client instance.

node_id is the internal ID that is used to identify the node when communicating with the server.

click()
Alias for `left_click`.

double_click()
Double clicks the current node, then waits for the page to fully load.

drag_to(*element*)
Drag the node to another one.

eval_script(*js*)
Evaluate arbitrary Javascript with the `node` variable bound to the current node.

exec_script(*js*)
Execute arbitrary Javascript with the `node` variable bound to the current node.

focus()
Puts the focus onto the current node, then waits for the page to fully load.

get_attr(*name*)
Returns the value of an attribute.

get_bool_attr(*name*)
Returns the value of a boolean HTML attribute like *checked* or *disabled*

get_node_factory()
Returns the associated node factory.

hover()
Hovers over the current node, then waits for the page to fully load.

is_attached()
Checks whether the current node is actually existing on the currently active web page.

is_checked()
is the `checked` attribute set for this node?

is_disabled()
is the `disabled` attribute set for this node?

is_multi_select()
is this node a multi-select?

is_selected()
is the `selected` attribute set for this node?

is_visible()
Checks whether the current node is visible.

left_click()
Left clicks the current node, then waits for the page to fully load.

path()
Returns an XPath expression that uniquely identifies the current node.

right_click()
Right clicks the current node, then waits for the page to fully load.

select_option()
Selects an option node.

set(*value*)
Sets the node content to the given value (e.g. for input fields).

set_attr(*name*, *value*)
Sets the value of an attribute.

submit()
Submits a form node, then waits for the page to completely load.

tag_name()
Returns the tag name of the current node.

text()
Returns the inner text (*not* HTML).

unselect_options()
Unselects an option node (only possible within a multi-select).

value()
Returns the node's value.

exception webkit_server.**NodeError**
Bases: exceptions.Exception

A problem occured within a Node instance method.

class webkit_server.**NodeFactory**(*client*)
Bases: object

Implements the default node factory.

client is the associated client instance.

class webkit_server.**SelectionMixin**
Bases: object

Implements a generic XPath selection for a class providing _get_xpath_ids, _get_css_ids and get_node_factory methods.

css(*css*)
Finds another node by a CSS selector relative to the current node.

xpath(*xpath*)
Finds another node by XPath originating at the current node.

class webkit_server.**Server**(*binary=None*)
Bases: object

Manages a Webkit server process. If *binary* is given, the specified webkit_server binary is used instead of the included one.

connect()
Returns a new socket connection to this server.

kill()
Kill the process.

class webkit_server.**ServerConnection**(*server=None*)
Bases: object

A connection to a Webkit server.

server is a server instance or *None* if a singleton server should be connected to (will be started if necessary).

issue_command(*cmd*, **args*)
Sends and receives a message to/from the server

class webkit_server.**SocketBuffer**(*f*)
Bases: object

A convenience class for buffered reads from a socket.

read(*n*)
Consume *n* characters from the stream.

read_line()
Consume one line from the stream.

exception webkit_server.**WebkitServerError**
Bases: exceptions.Exception

Raised when the Webkit server experiences an error.

webkit_server.**get_default_server**()
Returns a singleton Server instance (possibly after creating it, if it doesn't exist yet).

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

d

w