Raccy

Release 1.2.5

Daniel Afriyie

Oct 14, 2021

CONTENTS

Raccy is a multithreaded web scraping library based on selenium with built in Object Relational Mapper (ORM). It can be used for web automation, web scraping, and data mining. Currently the ORM feature supports only SQLite Database. Some of the features in this library is inspired by Django ORM and Scrapy.

CHAPTER

ONE

REQUIREMENTS

- Python 3.7+
- Works on Windows, Linux, and Mac

1.1 Contents:

1.1.1 Installation

Installing the latest version

Raccy requires python 3.7+. It is actually built with python 3.7. You can install the latest version hosted on PyPI with:

```
pip install raccy
```

Installing with git

The project is hosted at https://github.com/danielafriyie/raccy and can be installed using git:

```
git clone https://github.com/danielafriyie/raccy.git
cd raccy
python setup.py install
```

1.1.2 Tutorial

Raccy Tutorial

In this tutorial, we are going to scrape quotes.toscrape.com, a website that lists quotes from famous authors. We strongly recommend that you install **raccy** in a virtual environment to avoid conflict with your system packages. The source code for this tutorial is uploaded to github. You can find it from this link https://github.com/danielafriyie/raccy/blob/main/examples/quotes.py

This is the code we will use. Save it in a file called `quotes.py`:

```
from raccy import (
 model, UrlDownloaderWorker, CrawlerWorker, DatabaseWorker
 )
 from selenium import webdriver
```

(continues on next page)

(continued from previous page)

```
from shutil import which

config = model.Config()
config.DATABASE = model.SQLiteDatabase('quotes.sqlite3')


class Quote(model.Model):
    quote_id = model.PrimaryKeyField()
    quote = model.TextField()
    author = model.CharField(max_length=100)


class UrlDownloader(UrlDownloaderWorker):
    start_url = 'https://quotes.toscrape.com/page/1/'
    max_url_download = 10

    def job(self):
        url = self.driver.current_url
        self.url_queue.put(url)
        self.follow(xpath="//a[contains(text(), 'Next')]", callback=self.job)


class Crawler(CrawlerWorker):

    def parse(self, url):
        self.driver.get(url)
        quotes = self.driver.find_elements_by_xpath("//div[@class='quote']")
        for q in quotes:
            quote = q.find_element_by_xpath(".//span[@class='text']").text
            author = q.find_element_by_xpath(".//span/small").text

            data = {
                'quote': quote,
                'author': author
            }
            self.log.info(data)
            self.db_queue.put(data)


class Db(DatabaseWorker):

    def save(self, data):
        Quote.objects.create(**data)


def get_driver():
    driver_path = which('.\\chromedriver.exe')
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    options.add_argument("--start-maximized")
    driver = webdriver.Chrome(executable_path=driver_path, options=options)
    return driver
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    workers = []
    urldownloader = UrlDownloader(get_driver())
    urldownloader.start()
    workers.append(urldownloader)

    for _ in range(5):
        crawler = Crawler(get_driver())
        crawler.start()
        workers.append(crawler)

    db = Db()
    db.start()
    workers.append(db)

    for worker in workers:
        worker.join()

    print('Done scraping...........')
```

Now all you have to do is run the code above and you are done!

Diving into the code

Models

The models are designed is such a way that, the tables are created immediately you subclass the `model.Model` class without creating any object or instances or calling any create method. The tables will be created automatically when you run your code. The idea behind this is that, in web scraping, most of the time you'll be inserting data into a database. So instead of writing code to define your models and and also writing code to create them, you just define your models and start inserting data into them. Off course this behaviour can be turned off. You can read more in the API Documentation.

In our model defined above `Quote`, there are just three fields:

quote_id represents the primary key field for our table.

quote this field stores the actual quote that we will scrape.

author this field stores the name of the author who created the quote.

UrlDownloader

As you can see, this class subclass the `UrlDownloaderWorker` class. This class is responsible for downloading the urls of items, in this case quotes, that we will scrape. Let us take a look at the attributes and methods defined:

- *start_url:* this is the initial url our `UrlDownloader` will request from.
- *max_url_download:* this defines the maximum number of urls the `UrlDownloader` is supposed to donwload.
- *job:* this method is called to handle url extraction and also puts the extracted url into `ItemUrlQueue`

Crawler

This class subclass `CrawlerWorker` class. This class is responsible for fetching web pages of the items we want to scrape. In our case quotes. The class receives url from `ItemUrlQueue`, fetches the web page and scrape or extract data from it. Let us take a look at the methods defined:

- *parse:* this method is called to fetch web pages and scrape or extract data from them. The url parameter is the url received from `ItemUrlQueue`. The data is then put into `DatabaseQueue`.

Db

This class subclass `DatabaseWorker` class. This class is responsible for storing scraped data into persistent database. Let us take a look at some of the methods defined:

- *save:* this method is called to handle the process of storing scraped data into a database. The data parameter is the data received from `DatabaseQueue`.

1.1.3 Architecture Overview

UrlDownloaderWorker

Resonsible for downloading item(s) to be scraped urls and enqueue(s) them in **ItemUrlQueue**

ItemUrlQueue

Receives item urls from **UrlDownloaderWorker** and enqueues them for feeding them to **CrawlerWorker**

CrawlerWorker

Fetches item web pages and scrapes or extract data from them and enqueues the data in **DatabaseQueue**

DatabaseQueue

Receives scraped item data from **CrawlerWorker(s)** and enques them for feeding them to **DatabaseWorker**.

DatabaseWorker

Receives scraped data from **DatabaseQueue** and stores it in a persistent database.

1.1.4 API Documentation

This document specifies Raccy's APIs.

UrlDownloaderWorker API

class UrlDownloaderWorker (driver, *args, **kwargs):

Parameters

- **driver** - selenium webdriver object
- ***args** - arguments to pass to python threading.Thread class
- ****kwargs** - keyword arguments to to pass to python threading.Thread class

start_url - this is the initial url to make request from
url_queue - `ItemUrlQueue` object
mutex - python threading.Lock object
urls_scraped - total url downloaded
max_url_download - maximum number of urls to download
log - raccy.logger.logger.logger object
pre_job
This method is called before job method is called.
In case you want to do authentication or perform some action before doing the actual scraping, overwrite this method.
post_job
This method is called after job method is called, when all the scraping is done
wait (xpath, secs=5, condition=None, action=None)
Wrapper method acround selenium webdriver wait
follow (xpath=None, url=None, callback=None, *cbargs, **cbkwargs)
Follows the url or the button to click to go to the next page
job
This is where the actual scraping takes place.
close_driver
Calls driver.quit() on the selenium driver object

CrawlerWorker API

class CrawlerWorker (driver, *args, **kwargs):

Parameters

- **driver** - selenium webdriver object
- ***args** - arguments to pass to python threading.Thread class
- ****kwargs** - keyword arguments to to pass to python threading.Thread class

url_wait_timeout - how long to wait for urls from `ItemUrlQueue`
url_queue - ItemUrlQueue object
db_queue - DatabaseQueue object
log - raccy.logger.logger.logger object
pre_job
This method is called before parse method is called.

In case you want to do authentication or perform some action before doing the actual scraping, overwrite this method.

post_job

This method is called after parse method is called, when all the scraping is done

wait (xpath, secs=5, condition=None, action=None)

Wrapper method acround selenium webdriver wait

parse

This is where the actual scraping takes place.

close_driver

Calls driver.quit() on the selenium driver object

DatabaseWorker API

class DatabaseWorker:

wait_timeout - how long to wait for data from `DatabaseQueue`

db_queue - `DatabaseQueue` object

log - raccy.logger.logger.logger object

pre_job

This method is called before save method is called.

post_job

This method is called after save method is called.

save

This method is called to save data to a database

ORM API

class Config:

DATABASE

DBMAPPER

class PrimaryKeyField:

class CharField (max_length=None, null=True, unique=False, default=None):

class TextField (null=True, default=None):

class IntegerField (null=True, default=None):

class FloatField (null=True, default=None):

class BooleanField (null=True, default=None):

class DateField (null=True, default=None):

class DateTimeField (null=True, default=None):

class ForeignKeyField (model, on_field):

class Model:

class Meta:

abstract = False

table_name = None

create_table = True