FREE eBook

LEARNING scrapy

Free unaffiliated eBook created from
Stack Overflow contributors.

#scrapy

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: scrapy

It is an unofficial and free scrapy ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official scrapy.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with scrapy

Remarks

This section provides an overview of what scrapy is, and why a developer might want to use it.

It should also mention any large subjects within scrapy, and link out to the related topics. Since the Documentation for scrapy is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
1.1.2	2016-08-18

Examples

Installation of Scrapy

prerequisite of scrapy installation:

- Python 2.7 or above 3.3
- pip and setuptools Python packages.
- lxml
- OpenSSL.

You can install Scrapy using pip. To install using `pip` run:

```
pip install Scrapy
```

Platform specific installation

Anaconda

> *This is the recommended way to install Scrapy.*

If you already have installed Anaconda or Miniconda, the company Scrapinghub maintains official conda packages for Linux, Windows and OS X.

To install Scrapy using conda, run:

```
conda install -c scrapinghub scrapy
```

Ubuntu 9.10 or above

Use the official Ubuntu Packages, which already solve all dependencies for you and are continuously updated with the latest bug fixes.

If you prefer to build the python dependencies locally instead of relying on system packages you'll need to install their required non-python dependencies first:

```
sudo apt-get install python-dev python-pip libxml2-dev libxslt1-dev zlib1g-dev libffi-dev
libssl-dev
```

You can install Scrapy with `pip` after that:

```
pip install Scrapy
```

Archlinux

You can follow the generic instructions or install Scrapy from AUR Scrapy package:

```
yaourt -S scrapy
```

Windows

> Scrapy with Python 3 is not yet supported on Windows.

Follow This steps to install scrapy on windows:

- Install Python 2.7
- adjust PATH environment variable to include paths to the Python executable and additional scripts. The following paths need to be added to PATH:

  > C:\Python27;C:\Python27\Scripts;

- Install pywin32 from here
- let's install Scrapy:

  ```
  pip install Scrapy
  ```

Mac OS X

Building Scrapy's dependencies requires the presence of a C compiler and development headers. On OS X this is typically provided by Apple's Xcode development tools. To install the Xcode command line tools open a terminal window and run:

```
xcode-select --install
```

There's a known issue that prevents `pip` from updating system packages. This has to be addressed to successfully install Scrapy and its dependencies. Here are some proposed solutions:

- (Recommended) Don't use system python, install a new, updated version that doesn't conflict with the rest of your system. Here's how to do it using the homebrew package manager:
    - Install homebrew following the instructions in http://brew.sh/
    - Update your `PATH` variable to state that homebrew packages should be used before system packages (Change `.bashrc` to `.zshrc` accordantly if you're using zsh as default shell):

      ```
      echo "export PATH=/usr/local/bin:/usr/local/sbin:$PATH" >> ~/.bashrc
      ```

    - Reload `.bashrc` to ensure the changes have taken place:

      ```
      source ~/.bashrc
      ```

    - Install python:

      ```
      brew install python
      ```

    - Latest versions of python have `pip` bundled with them so you won't need to install it separately. If this is not the case, upgrade python:

      ```
      brew update; brew upgrade python
      ```

- (Optional) Install Scrapy inside an isolated python environment.

  This method is a workaround for the above OS X issue, but it's an overall good practice for managing dependencies and can complement the first method.

  virtualenv is a tool you can use to create virtual environments in python. We recommended reading a tutorial like http://docs.python-guide.org/en/latest/dev/virtualenvs/ to get started.

After any of these workarounds you should be able to install Scrapy:

```
pip install Scrapy
```

Creating a project

Before starting work with scrapy you have to start a project where you want to store your code. Enter the directory and run this code

```
scrapy startproject helloProject
```

The third part of this code is project name. This code will create a "helloProject" directory with the

following contents:

```
helloProject/
    scrapy.cfg            # deploy configuration file

    helloProject/         # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

Read Getting started with scrapy online: https://riptutorial.com/scrapy/topic/2099/getting-started-with-scrapy

Chapter 2: Connecting scrapy to MySQL

Examples

Connecting scrapy to MySQL (Windows 8 pro 64-bit, python 2.7, scrapy v 1.2)

The following example is tested on **Windows 8 pro 64-bit** operating system with **python 2.7** and **scrapy v 1.2**. Let assume that we have already installed the scrapy framework.

MySQL database that we will use in the following tutorial

```
CREATE TABLE IF NOT EXISTS `scrapy_items` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `quote` varchar(255) NOT NULL,
  `author` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `scrapy_items` (`id`, `quote`, `author`)
VALUES (1, 'The world as we have created it is a process of our thinking. It cannot be changed
without changing our thinking.', 'Albert Einstein');
```

Installation MySQL driver

1. Download driver mysql-connector-python-2.2.1.zip OR MySQL-python-1.2.5.zip (md5)
2. Extract zip into a file e.g **C:\mysql-connector**
3. Open **cmd** go to the **C:\mysql-connector** where **setup.py** file will be located and run **python setup.py install**
4. Copy and run the following **example.py**

```
from __future__ import print_function
import mysql.connector
from mysql.connector import errorcode

class MysqlTest():
    table = 'scrapy_items'
    conf = {
        'host': '127.0.0.1',
        'user': 'root',
        'password': '',
        'database': 'test',
        'raise_on_warnings': True
    }

    def __init__(self, **kwargs):
        self.cnx = self.mysql_connect()

    def mysql_connect(self):
        try:
            return mysql.connector.connect(**self.conf)
        except mysql.connector.Error as err:
            if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
```

```
                print("Something is wrong with your user name or password")
            elif err.errno == errorcode.ER_BAD_DB_ERROR:
                print("Database does not exist")
            else:
                print(err)

    def select_item(self):
        cursor = self.cnx.cursor()
        select_query = "SELECT * FROM " + self.table

        cursor.execute(select_query)
        for row in cursor.fetchall():
            print(row)

        cursor.close()
        self.cnx.close()

def main():
    mysql = MysqlTest()
    mysql.select_item()

if __name__ == "__main__" : main()
```

Connect Scrapy to MySQL

First create a new scrapy project by running the following command

```
scrapy startproject tutorial
```

This will create a tutorial directory with the following contents:

```
tutorial/
    scrapy.cfg            # deploy configuration file

    tutorial/             # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

This is the code for our first Spider. Save it in a file named **quotes_spider.py** under the **tutorial/spiders** directory in your project.

Our first Spider

```
import scrapy
from scrapy.loader import ItemLoader
from tutorial.items import TutorialItem
```

```
class QuotesSpider(scrapy.Spider):
    name = "quotes"

    def start_requests(self):
        urls = ['http://quotes.toscrape.com/page/1/']
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        boxes = response.css('div[class="quote"]')
        for box in boxes:
            item = ItemLoader(item=TutorialItem())
            quote = box.css('span[class="text"]::text').extract_first()
            author = box.css('small[class="author"]::text').extract_first()
            item.add_value('quote', quote.encode('ascii', 'ignore'))
            item.add_value('author', author.encode('ascii', 'ignore'))
            yield item.load_item()
```

Scrapy Item Class

To define common output data format Scrapy provides the **Item** class. **Item** objects are simple containers used to collect the scraped data and specify metadata for the field. They provide a **dictionary-like** API with a convenient syntax for declaring their available fields. For detail click me

```
import scrapy
from scrapy.loader.processors import TakeFirst

class TutorialItem(scrapy.Item):
    # define the fields for your item here like:
    quote = scrapy.Field(output_processor=TakeFirst(),)
    author = scrapy.Field(output_processor=TakeFirst(),)
```

Scrapy Pipeline

After an item has been scraped by a spider, it is sent to the Item Pipeline which processes it through several components that are executed sequentially and this is the place where we save our scraped data into database. For detail click me

> **Note**: Don't forget to add your pipeline to the **ITEM_PIPELINES** setting located in **tutorial/tutorial/settings.py** file.

```
from __future__ import print_function
import mysql.connector
from mysql.connector import errorcode

class TutorialPipeline(object):
    table = 'scrapy_items'
    conf = {
        'host': '127.0.0.1',
        'user': 'root',
        'password': '',
        'database': 'sandbox',
        'raise_on_warnings': True
    }
```

```
    def __init__(self, **kwargs):
        self.cnx = self.mysql_connect()

    def open_spider(self, spider):
        print("spider open")

    def process_item(self, item, spider):
        print("Saving item into db ...")
        self.save(dict(item))
        return item

    def close_spider(self, spider):
        self.mysql_close()

    def mysql_connect(self):
        try:
            return mysql.connector.connect(**self.conf)
        except mysql.connector.Error as err:
            if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
                print("Something is wrong with your user name or password")
            elif err.errno == errorcode.ER_BAD_DB_ERROR:
                print("Database does not exist")
            else:
                print(err)


    def save(self, row):
        cursor = self.cnx.cursor()
        create_query = ("INSERT INTO " + self.table +
            "(quote, author) "
            "VALUES (%(quote)s, %(author)s)")

        # Insert new row
        cursor.execute(create_query, row)
        lastRecordId = cursor.lastrowid

        # Make sure data is committed to the database
        self.cnx.commit()
        cursor.close()
        print("Item saved with ID: {}" . format(lastRecordId))

    def mysql_close(self):
        self.cnx.close()
```

Ref: https://doc.scrapy.org/en/latest/index.html

Connecting and bulk-inserting to MySQL in Scrapy using MySQLDB module - Python 2.7

This example demonstrate how to dynamically insert data into MySQL using Python Scrapy.

You do not need to edit `pipelines.py` file for any project.

This example can be used for all your project.

Just `yield you_data_dictionary` from your `Spider` and inside `pipelines.py` a query will be created automatically.

Rows are inserted in bulk using bulk insert statement.

MUST READ:

1. Keys of you `you_data_dictionary` that you are `yield`ing from Spider must be same as your column names of database table.
2. Table must be created before you run your code.
3. Notice `if len(self.items) >= 50` line, you can change `50` to any integer.

settings.py

```
DB_CREDS = {
    'host':'localhost',
    'user':'root',
    'pass':'password',
    'db':'db_name'
}
```

your_project_folder/spiders/spider_file.py

```
from scrapy.utils.project import get_project_settings
        def __init__(self, *args, **kwargs):
                self.connectDB()

        def connectDB(self):

                self.conn = MySQLdb.connect(user=DB_CREDS['user'], passwd=DB_CREDS['pass'],
db=DB_CREDS['db'], host=DB_CREDS['host'], charset="utf8", use_unicode=True)

                self.cursor = MySQLdb.cursors.DictCursor(self.conn)

                self.conn.autocommit(True)
```

your_project_folder/pipelines.py

```
# -*- coding: utf-8 -*-
import logging
from scrapy import signals

class MyPipeline(object):

    def __init__(self):
        self.items=[]

    def process_item(self, item, spider):

                self.placeholders = ', '.join(['%s'] * len(item))
                self.columns = ', '.join(item.keys())
                self.query = "INSERT INTO %s ( %s ) VALUES ( %s )" % ("table_name",
self.columns, self.placeholders)

        self.items.extend([item.values()])

        if len(self.items) >= 50:
```

```
            try:
                spider.cursor.executemany(self.query, self.items)
                self.items = []
            except Exception as e:
                if 'MySQL server has gone away' in str(e):
                    spider.connectDB()
                    spider.cursor.executemany(self.query, self.items)
                    self.items = []
                    else:
                        raise e
        return item



        def close_spider(self, spider):
            try:
                    spider.cursor.executemany(self.query, self.items)
                    self.items = []
            except Exception as e:
                    if 'MySQL server has gone away' in str(e):
                            spider.connectDB()
                            spider.cursor.executemany(self.query, self.items)
                            self.items = []
                    else:
                            raise e
```

Read Connecting scrapy to MySQL online: https://riptutorial.com/scrapy/topic/7925/connecting-scrapy-to-mysql

Chapter 3: Item Pipeline

Introduction

Way to process every item that Scrapy outputs.

An Item Pipeline is a python class that overrides some specific methods and needs to be activated on the `settings` of the scrapy project.

Examples

Creating your own Pipeline

When creating a scrapy project with `scrapy startproject myproject`, you'll find a `pipelines.py` file already available for creating your own pipelines. It isn't mandatory to create your pipelines in this file, but it would be good practice. We'll be explaining how to create a pipeline using the `pipelines.py` file:

pipelines.py

```
class MyPipeline(object):
    def process_item(self, item, spider):
        # process your `item` here
        return item
```

Now to enable it you need to specify it is going to be used in your settings. Go to your `settings.py` file and search (or add) the `ITEM_PIPELINES` variable. Update it with the path to your pipeline class and its priority over other pipelines:

settings.py

```
ITEM_PIPELINES = {
    'myproject.pipelines.MyPipeline': 300,
}
```

Now every item that your spider returns, will go through this pipeline.

Creating a dynamic pipeline in Python Scrapy

Enable pipelines in your **`settings.py`**

```
ITEM_PIPELINES = {
    'project_folder.pipelines.MyPipeline': 100
}
```

Then write this code in **`items.py`**

```
# -*- coding: utf-8 -*-
from scrapy import Item, Field
from collections import OrderedDict

class DynamicItem(Item):
    def __setitem__(self, key, value):
        self._values[key] = value
        self.fields[key] = {}
```

Then in your `project_folder/spiders/spider_file.py

```
from project_folder.items import DynamicItem
       def parse(self, response):
               # create an ordered dictionary
               data = OrderedDict()
               data['first'] = ...
               data['second'] = ...
               data['third'] = ...
               .
               .
               .
               # create dictionary as long as you need

               # now unpack dictionary
               yield DynamicItem( **data )

               # above line is same as this line
               yield DynamicItem( first = data['first'], second = data['second'], third =
data['third'])
```

What are benefits of this code?

No need to create define each item in `items.py` one by one.

Read Item Pipeline online: https://riptutorial.com/scrapy/topic/8589/item-pipeline

Credits

S. No	Chapters	Contributors
1	Getting started with scrapy	Community, neverlastn, Razik, Roman Zaitsev, sudo bangbang
2	Connecting scrapy to MySQL	MasoodUrRehman, Umair, winseybash
3	Item Pipeline	eLRuLL, Umair