fulmar Documentation

Release master

November 04, 2016

Contents

Fulmar is a distributed crawler system.

By using non-blocking network I/O, Fulmar can handle hundreds of open connections at the same time. You can extract the data you need from websites. In a fast, simple way.

Some features you may want to know:

- Write script in Python
- Task crontab, priority
- Cookie persistence
- Use Redis as message queue
- Use MongoDB as default database at present
- Support rate limitation of requests for a certain website
- Distributed architecture
- Crawl Javascript pages

CHAPTER 1

Quick links

- Source (github)
- Wiki

Script example

```
from fulmar.base_spider import BaseSpider

class Handler(BaseSpider):

   def on_start(self):
      self.crawl('http://www.baidu.com/', callback=self.detail_page)

   def parse_and_save(self, response):
      return {
         "url": response.url,
         "title": response.page_lxml.xpath('//title/text()')[0]}
```

You can save above code in a new file called `baidu_spider.py` and run command:

```
fulmar start_project baidu_spider.py
```

If you have installed *redis*, you will get:

```
Successfully start the project, project name: "baidu_spider".
```

Finally, start Fulmar:

```
fulmar all
```

CHAPTER 3

Installation

Automatic installation:

```
pip install fulmar
```

Fulmar is listed in PyPI and can be installed with `pip` or `easy_install`.

Manual installation: Download tarball, then:

```
tar xvzf fulmar-master.tar.gz
cd fulmar-master
python setup.py build
sudo python setup.py install
```

The Fulmar source code is hosted on GitHub.

Prerequisites: Fulmar runs on Python 2.7, and 3.3+ For Python 2, version 2.7.9 or newer is *strongly* recommended for the improved SSL support.

CHAPTER 4

Documentation

This documentation is also available in PDF and Epub formats.

4.1 Quickstart

4.1.1 Installation

- *pip install fulmar*

Note: Redis is necessary, so make sure you have installed it.

Note: If you want to save the data you extracted from websites, please install MongoDB.

Please install PhantomJS if needed: http://phantomjs.org/build.html

Note: *PhantomJS* will be enabled only if it is excutable in the *PATH* or in the System Environment.

4.1.2 Run

- `fulmar all`

Note: `fulmar all` command is running fulmar all in one, which running components in threads or subprocesses. For production environment, please refer to Deployment.

4.1.3 Your First Spider

```
from fulmar.base_spider import BaseSpider

class Handler(BaseSpider):

   def on_start(self):
      self.crawl('http://www.baidu.com/', callback=self.parse_and_save)

   def parse_and_save(self, response):
      return {
         "url": response.url,
         "title": response.page_lxml.xpath('//title/text()')[0]}
```

You can save above code in a new file called *baidu_spider.py* and run command in a new console:

```
fulmar start_project baidu_spider.py
```

If you have installed *redis*, you will get:

```
Successfully start the project, project name: `baidu_spider`.
```

In the example:

on_start(self)

It is the entry point of the spider.

self.crawl(url, callback=self.parse_and_save)

It is the most important API here. It will add a new task to be crawled.

parse_and_save(self, response)

It get a Response object. response.page_lxml is a lxml.html document_fromstring object which has `xpath` API to select elements to be extracted.

It return a *dict* object as result. The result will be captured into `resultdb` by default. You can override `on_result(self, result)` method to manage the result yourself.

More details you need to know:

CrawlRate

It is a decorator for the `Handler` class. It is used for limiting the crawl rate.

You can use it just like:

```
from fulmar.base_spider import BaseSpider, CrawlRate

@CrawlRate(request_number=1, time_period=2)
class Handler(BaseSpider):

   def on_start(self):
      self.crawl('http://www.baidu.com/', callback=self.parse_and_save)

   def parse_and_save(self, response):
      return {
         "url": response.url,
         "title": response.page_lxml.xpath('//title/text()')[0]}
```

It means you can only send `requests_number` requests during `time_period` seconds. Note that this rate limitation is used for a Worker.

So if you start *fulmar* with `n` workers, you actually send `requests_number * n` requests during `time_period` seconds.

4.2 Opitions

Fulmar is easy to use.

Note: *Redis* is necessary, so make sure you have installed it.

4.2.1 –help

You can get `help`, just run:

```
fulmar --help
```

You will see:

```
Usage: fulmar [OPTIONS] COMMAND [ARGS]...

A  crawler system.

Options:
  -c, --config TEXT        a yaml file with default config.  [default:
                           /fulmar/fulmar/config.yml]
  --redis TEXT             redis address, e.g, 'redis://127.0.0.1:6379/0'.
  --mongodb TEXT           mongodb address, e.g, 'mongodb://localhost:27017/'.
  --phantomjs-proxy TEXT phantomjs proxy ip:port.
  --logging-config TEXT  logging config file for built-in python logging
                           module  [default: /fulmar/fulmar/logging.conf]

  --version                Show the version and exit.
  --help                   Show this message and exit.

 Commands:
   all              Start scheduler and worker, also run...
   crontab          Crontab infos and operations.
   delete_project   Delete a project.
   phantomjs        Run phantomjs if phantomjs is installed.
   scheduler        Run Scheduler.
   show_projects    Show projects.
   start_project    Start a project.
   stop_project     Stop a project.
   update_project   Update a project.
   worker           Run Worker.
```

4.2.2 –config

Config file is a YAML file with config values for global options or subcommands. Fulmar has a default config file, the content is:

```
redis:
    url: redis://127.0.0.1:6379/0
mongodb:
    url: mongodb://localhost:27017/
worker:
    async: true
    poolsize: 300
    timeout: 180
```

If you run fulmar without any paramtets or config file, fulmar will use this default configuration. You can write your own config file, and use it just like:

```
fulmar --config=your-config-file all
```

4.2.3 –redis

Redis address. You can run fulmar just like:

```
fulmar --redis=redis://127.0.0.1:6379/0 all
```

4.2.4 –mongodb

MongoDB address.

4.2.5 –phantomjs-proxy

phantomjs proxy ip:port. If you set it, it means you have already run phantomjs. So fulmar will not try to run a new phantomjs, instead just use this one.

4.2.6 –logging-config

Log config file. Fulmar use logging. If you want to change the default log behavior, you can write you own log file, reference: configuration-file-format

4.2.7 –version

Show fulmar version.

4.3 Service

Note: *Redis* is necessary, so make sure you have installed it.

More precisely, fulmar is a sevice which can run all the time. Fulmar mainly has three parts, scheduler, worker and phantomjs. You can start them all in one, or run them separately.

4.3.1 all

Run fulmar all in one. Start scheduler and worker. If phantomjs is installed and global opitions `phantomjs-proxy` isn't provided, phantomjs will get started too.

4.3.2 phantomjs

Run phantomjs when phantomjs is installed and global opitions `phantomjs-proxy` isn't provided.

4.3.3 scheduler

Run Scheduler. Note that you should only start one scheduler.

4.3.4 worker

Run worker.

You can get help, just run:

`fulmar worker --help`

You will see:

```
Usage: fulmar worker [OPTIONS]

Run Worker.

Options:
  --poolsize INTEGER  pool size
  --user-agent TEXT   user agent
  --timeout INTEGER   default request timeout
  --help              Show this message and exit.
```

–poolsize

The maximum number of simultaneous fetch operations that can execute in parallel. Defaults to 300.

–timeout

The request timeout. Defaults to 180s.

4.4 Project

Note: *Redis* is necessary, so make sure you have installed it.

4.4.1 start_project

```
from fulmar.base_spider import BaseSpider

class Handler(BaseSpider):

   def on_start(self):
      self.crawl('http://www.baidu.com/', callback=self.detail_page)

   def parse_and_save(self, response):
      return {
         "url": response.url,
         "title": response.page_lxml.xpath('//title/text()')[0]}
```

You can save above code in a new file called `baidu_spider.py` and run command in a new console:

```
fulmar start_project baidu_spider.py
```

you will get:

```
Successfully start the project, project name: `baidu_spider`.
```

We created a new project called `baidu_spider` and started it.

As you can see, project name is from the script file name. So you should keep the file name unique.

Note: In fact `start_project` just put a new task into new task queue. So, If your fulmar service is not running, this project will not start until you start fulmar.

4.4.2 stop_project

Run command:

```
fulmar stop_project baidu_spider
```

you will get:

```
Successfully stop project: "baidu_spider".
```

Stop a project. It means any tasks whose project name is `baidu_spider` will be stop immediately. At the same time, any new tasks in `fulmar crontab` will not run.

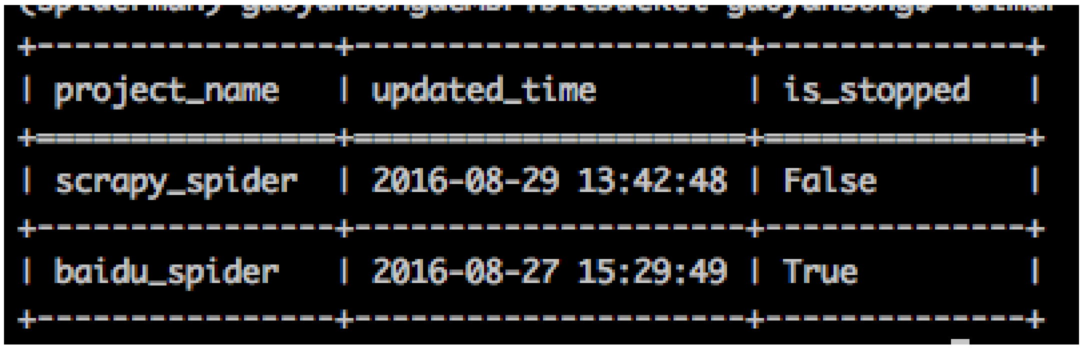
4.4.3 show_projects

Show projects status which are in projectdb.

Run command:

```
fulmar show_projects
```

you will get:



```
+----------------+---------------------+---------------+
| project_name   | updated_time        | is_stopped    |
+================+=====================+===============+
| scrapy_spider  | 2016-08-29 13:42:48 | False         |
+----------------+---------------------+---------------+
| baidu_spider   | 2016-08-27 15:29:49 | True          |
+----------------+---------------------+---------------+
```

4.4.4 delete_project

Delete a project. This will only delete a project from projectdb.

4.5 Crontab

4.5.1 Start

If you want to run some tasks periodically or run tasks at the absolute time in the future, you can use these parameters in `self.crawl()`:

`crawl_at`: The absolute time to start the crawl. It must be a timestamp.

`crawl_later`: Starts the crawl after `crawl_later` seconds.

crawl_period: Schedules the request to be called periodically. The crawl is called every `crawl_period` seconds.

For example:

```
from fulmar.base_spider import BaseSpider

class Handler(BaseSpider):

    base_url = 'http://doc.scrapy.org/en/latest/'

    def on_start(self):
        self.crawl(Handler.base_url, callback=self.save, crawl_period=60*60)

    def save(self, response):
        return {
            'content': response.content,
            'title': response.page_lxml.xpath('//title/text()')[0]
        }
```

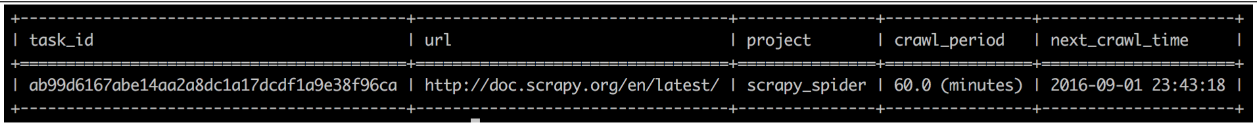
Now we save it to a file called `scrapy_spider` and run command:

```
fulmar start_project scrapy_spider.py
```

We satrtted a project called `scrapy_spider` and it will run every one hour.

It's convenient to see it by:

```
fulmar crontab
```



```
+----------------------------------+----------------------------------+---------------+----------------+---------------------+
| task_id                          | url                              | project       | crawl_period   | next_crawl_time     |
+==================================+==================================+===============+================+=====================+
| ab99d6167abe14aa2a8dc1a17dcdf1a9e38f96ca | http://doc.scrapy.org/en/latest/ | scrapy_spider | 60.0 (minutes) | 2016-09-01 23:43:18 |
+----------------------------------+----------------------------------+---------------+----------------+---------------------+
```

4.5.2 –help

You can get `help`, just run:

`fulmar crontab --help`

You will see:

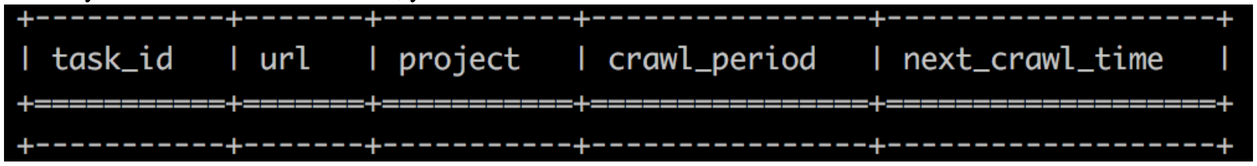
```
Usage: fulmar crontab [OPTIONS]

  Crontab infos and operations.

Options:
  -d, --delete TEXT  Delete a cron task. Here use taskid, e.g, -d taskid
  -v, --verbose      Verbose mode. Show more information about this crontab.
  --help             Show this message and exit.
```

4.5.3 –delete/-d

Delete a cron task. Here use taskid to represents a cron task. You can delete a task which you put just now:

```
fulmar crontab --delete=ab99d6167abe14aa2a8dc1a17dcdf1a9e38f96ca
```

Now if you run `fulmar crontab`, you will see:



```
+-----------+-------+-----------+---------------+-------------------+
| task_id   | url   | project   | crawl_period  | next_crawl_time   |
+===========+=======+===========+===============+===================+
+-----------+-------+-----------+---------------+-------------------+
```

The task has been deleted successfully.

4.5.4 –vorbose/-v

Verbose mode. Show more information about this crontab.

4.6 API Index

4.6.1 BaseSpider

It's the main class.

You should use it in your script file like this:

```
from fulmar.base_spider import BaseSpider
```

Then you should create a new class which inherit `BaseSpider` just like:

```
class Handler(BaseSpider)
```

Note that the name of the class don't have to be `Handler`.

4.6.2 CrawlRate

It is used for limiting the crawl rate. It is a decorator for the `Handler` class.

You can use it just like:

```
from fulmar.base_spider import BaseSpider, CrawlRate

@CrawlRate(request_number=1, time_period=2)
class Handler(BaseSpider):

   def on_start(self):
      self.crawl('http://www.baidu.com/', callback=self.parse_and_save)

   def parse_and_save(self, response):
      return {
         "url": response.url,
         "title": response.page_lxml.xpath('//title/text()')[0]}
```

It means you can only send `requests_number` requests during `time_period` seconds. Note that this rate limitation is used for one Worker.

So if you start *fulmar* with `n` workers, you actually send `requests_number * n` requests during `time_period` seconds.

4.6.3 def start(self)

In the class `Handler`, you should define a method called `start`. It's the entrance of the whole crawl.

4.6.4 def crawl(self, url, **kwargs)

It's used to define a new crawl task. You can use it like:

```
self.crawl('http://www.baidu.com/', callback=self.detail_page)
```

There are many parameters can be used.

url

URL for the new request.

method

method for the new request. Defaults to `GET`. (optional) Dictionary or bytes to be sent in the query string for the request.

data

(optional) Dictionary or bytes to be send in the body of request.

headers

(optional) Dictionary of HTTP Headers to be send with the request.

cookies

(optional) Dictionary to be send with the request.

cookie_persistence

Previous request and response's cookies will persist next request for the same website. Defaults to `True`. Type: `bool`.

timeout

(optional) How long to wait for the server to send data before giving up. Type: `float`.

priority

The bigger, the higher priority of the request. Type: `int`.

callback

The method to parse the response.

callback_args

The additional args to the callback. Type: `list`.

callback_kwargs

The additional kwargs to the callback. Type: `dict`.

taskid

(optional) Unique id to identify the task. Default is the sha1 check code of the URL. But it won't be unique when you request the same url with different post params.

crawl_at

The time to start the rquest. It must be a timestamp. Type: `Int` or `Float`.

crawl_later

Starts the request after `crawl_later` seconds.

crawl_period

Schedules the request to be called periodically. The request is called every `crawl_period` seconds.

crawl_rate

This should be a dict Which contain `request_number` and `time_period`. Note that the `time_period` is given in seconds. If you don't set `time_period`, the default is 1. E.g.,

```
crawl_rate={'request_number': 10, 'time_period': 2}
```

Which means you can crawl the url at most 10 times every 2 seconds.

Type: dict.

allow_redirects

(optional) Boolean. Defaults to `True`. Type: `bool`.

proxy_host

(optional) HTTP proxy hostname. To use proxies, proxy_host and proxy_port must be set; proxy_username and proxy_password are optional. Type: `string`.

proxy_port

(optional) HTTP proxy port. Type: `Int`.

proxy_username

(optional) HTTP proxy username. Type: `string`.

proxy_password

(optional) HTTP proxy password. Type: `string`.

fetch_type

(optional) Set to `js` to enable JavaScript fetcher. Defaults to `None`.

js_script

(optional) JavaScript run before or after page loaded, should been wrapped by a function like `function() { document.write("Hello World !"); }`.

js_run_at

(optional) Run JavaScript specified via js_script at `document-start` or `document-end`. Defaults to `document-end`.

js_viewport_width

(optional) Set the size of the viewport for the JavaScript fetcher of the layout process.

js_viewport_height

(optional) Set the size of the viewport for the JavaScript fetcher of the layout process.

load_images

(optional) Load images when JavaScript fetcher enabled. Defaults to `False`.

validate_cert

(optional) For HTTPS requests, validate the server's certificate? Defaults to `True`.