scrapyrt Documentation

Release 0.10

scrapinghub

May 09, 2018

Contents

HTTP server which provides API for scheduling Scrapy spiders and making requests with spiders.

CHAPTER 1

Installation

To install Scrapyrt:

```
pip install scrapyrt
```

Now you can run Scrapyrt from within Scrapy project by just typing:

```
scrapyrt
```

in Scrapy project directory. This should start server on port 9080. You may change the port server will listen to using `-p` option (see *Command line arguments*):

```
scrapyrt -p 9081
```

Scrapyrt will look for `scrapy.cfg` file to determine your project settings, and will raise error if it won't find one. Note that you need to have all your project requirements installed.

Pay attention to Scrapy version you're using in your spiders. Scrapyrt makes use of recent improvements in Scrapy Crawler interface that are not present in old Scrapy versions. Look closely at `requirements.txt` of Scrapyrt and install most recent development Scrapy version if possible. Unfortunately we are unable to support old Scrapy versions.

If you would like to play with source code and possibly contribute to the project, you can install Scrapyrt in 'dev' mode:

```
python setup.py develop
```

with this mode of installation changes you made to Scrapyrt source will be reflected when you run `scrapyrt` command.

In production you can run Scrapyrt from docker image provided by Scrapinghub. You only need to do following things:

```
docker pull scrapinghub/scrapyrt
```

This will download Scrapyrt Docker image for you. Next step you need to run this image. Remember about providing proper port and project directory. Project directory from host machine must be mounted in directory /scrapyrt/project on guest. Following command will launch Scrapyrt forwarding port 9080 from guest to host, in demonized mode, with project directory in directory /home/user/quotesbot:

```
docker run -p 9080:9080 -tid -v /home/user/quotesbot:/scrapyrt/project scrapinghub/
↪scrapyrt
```

If you'd like to test if your virtual container is running just run:

```
docker ps
```

this command should return container_id, image etc. Testing with curl:

```
curl -v "http://localhost:9080/crawl.json?url=http://example.com&spider_name=toscrape-
↪css" | jq
```

should return expected response.

CHAPTER 2

Scrapyrt HTTP API

Scrapyrt supports endpoint `/crawl.json` that can be requested with two methods.

2.1 GET

2.1.1 Arguments

Currently it accepts following arguments:

spider_name

- type: string
- required

Name of the spider to be scheduled. If spider is not found api will return 404.

url

- type: string
- required if start_requests not enabled

Absolute URL to send request to. URL should be urlencoded so that querystring from url will not interfere with api parameters.

By default API will crawl this url and won't execute any other requests. Most importantly it will not execute `start_requests` and spider will not visit urls defined in `start_urls` spider attribute. There will be only one single request scheduled in API - request for resource identified by url argument.

If you want to execute request pass start_requests argument.

callback

- type: string
- optional

Should exist as method of scheduled spider, does not need to contain self. If not passed or not found on spider default callback parse will be used.

max_requests

- type: integer
- optional

Maximum amount of requests spider can generate. E.g. if it is set to `1` spider will only schedule one single request, other requests generated by spider (for example in callback, following links in first response) will be ignored. If your spider generates many requests in callback and you don't want to wait forever for it to finish you should probably pass it.

start_requests

- type: boolean
- optional

Whether spider should execute `Scrapy.Spider.start_requests` method. `start_requests` are executed by default when you run Scrapy Spider normally without ScrapyRT, but this method is NOT executed in API by default. By default we assume that spider is expected to crawl ONLY url provided in parameters without making any requests to `start_urls` defined in `Spider` class. start_requests argument overrides this behavior. If this argument is present API will execute start_requests Spider method.

If required parameters are missing api will return 400 Bad Request with hopefully helpful error message.

2.1.2 Examples

To run sample dmoz spider from Scrapy educational dirbot project parsing page about Ada programming language:

```
curl "http://localhost:9080/crawl.json?spider_name=dmoz&url=http://www.dmoz.org/
↪Computers/Programming/Languages/Ada/"
```

To run same spider only allowing one request and parsing url with callback `parse_foo`:

```
curl "http://localhost:9080/crawl.json?spider_name=dmoz&url=http://www.dmoz.org/
↪Computers/Programming/Languages/Ada/&callback=parse_foo&max_requests=1"
```

2.2 POST

Request body must contain valid JSON with information about request to be scheduled with spider and spider name. All positional and keyword arguments for Scrapy Request should be placed in request JSON key. Sample JSON:

```
{
    "request": {
        "url": "http://www.target.com/p/-/A-13631176",
        "callback": "parse_product",
        "dont_filter": "True"
    },
    "spider_name": "target.com_products"
}
```

Slighty more complicated JSON:

```
{
    "request": {
        "url": "http://www.target.com/p/-/A-13631176",
        "meta": {
            "category": "some category",
            "item": {
                "discovery_item_id": "999"
            }
        },
        "callback": "parse_product",
        "dont_filter": "True",
        "cookies": {
            "foo": "bar"
        }
    },
    "spider_name": "target.com_products"
}
```

2.2.1 Arguments

JSON in POST body must have following keys:

spider_name

- type: string
- required

Name of the spider to be scheduled. If spider is not found api will return 404.

max_requests

- type: integer
- optional

Maximal amount of requests spider can generate.

request

- type: JSON object
- required

Should be valid JSON containing arguments to Scrapy request object that will be created and scheduled with spider.

request JSON object must contain following keys:

url

- type: string
- required

It can contain all keyword arguments supported by Scrapy Request class.

If required parameters are missing api will return 400 Bad Request with hopefully helpful error message.

2.2.2 Examples

To schedule spider dmoz with sample url using POST handler:

```
curl localhost:9080/crawl.json \
    -d '{"request":{"url":"http://www.dmoz.org/Computers/Programming/Languages/Awk/"},
↪ "spider_name": "dmoz"}'
```

to schedule same spider with some meta that will be passed to spider request:

```
curl localhost:9080/crawl.json \
    -d '{"request":{"url":"http://www.dmoz.org/Computers/Programming/Languages/Awk/",
↪"meta": {"alfa":"omega"}}, "spider_name": "dmoz"}'
```

2.3 Response

`/crawl.json` returns JSON object. Depending on whether request was successful or not fields in json object can vary.

2.3.1 Success response

JSON response for success has following keys:

status Success response always have status "ok".

spider_name Spider name from request.

stats Scrapy stats from finished job.

items List of scraped items.

items_dropped List of dropped items.

errors (optional) Contains list of strings with crawl errors tracebacks. Available only if *DEBUG* settings is set to `True`.

Example:

```
$ curl "http://localhost:9080/crawl.json?spider_name=dmoz&url=http://www.dmoz.org/
↪Computers/Programming/Languages/Ada/"
{
    "status": "ok"
    "spider_name": "dmoz",
    "stats": {
        "start_time": "2014-12-29 16:04:15",
        "finish_time": "2014-12-29 16:04:16",
        "finish_reason": "finished",
        "downloader/response_status_count/200": 1,
        "downloader/response_count": 1,
        "downloader/response_bytes": 8494,
        "downloader/request_method_count/GET": 1,
        "downloader/request_count": 1,
        "downloader/request_bytes": 247,
        "item_scraped_count": 16,
        "log_count/DEBUG": 17,
        "log_count/INFO": 4,
```

(continues on next page)

(continued from previous page)

```
        "response_received_count": 1,
        "scheduler/dequeued": 1,
        "scheduler/dequeued/memory": 1,
        "scheduler/enqueued": 1,
        "scheduler/enqueued/memory": 1
    },
    "items": [
        {
            "description": ...,
            "name": ...,
            "url": ...
        },
        ...
    ],
    "items_dropped": [],
}
```

2.3.2 Error response

JSON error response has following keys:

status Error response always have status “error”.

code Duplicates HTTP response code.

message Error message with some explanation why request failed.

Example:

```
$ curl "http://localhost:9080/crawl.json?spider_name=foo&url=http://www.dmoz.org/
↪Computers/Programming/Languages/Ada/"
{
    "status": "error"
    "code": 404,
    "message": "Spider not found: foo",
}
```

CHAPTER 3

Tweaking spiders for realtime

If you have some standard values you would like to add to all requests generated from realtime api and you don't want to pass them in each GET request sent to api you can add a method `modify_realtime_request` to your spider, this method should accept request and return modified request you would like to send. API will execute this method, modify request and issue modified request.

For example:

```
class SpiderName(Spider):
    name = "some_spider"

    def parse(self, response):
        pass

    def modify_realtime_request(self, request):
        request.meta["dont_redirect"] = True
        return request
```

One more example (don't forget to import random):

```
class SpiderName(Spider):
    name = "some_other_spider"

    def parse(self, response):
        pass

    def modify_realtime_request(self, request):
        UA = [
            'Mozilla/5.0 (Windows NT 5.1; rv:31.0) Gecko/20100101 Firefox/31.0',
            'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_0) AppleWebKit/537.36␣
↪(KHTML, like Gecko) Chrome/37.0.2062.94 Safari/537.36',
        ]
        request.headers["User-Agent"] = random.choice(UA)
        return request
```

CHAPTER 4

Command line arguments

Use `scrapyrt -h` to get help on command line options:

```
$ scrapyrt -h
usage: scrapyrt [-h] [-p PORT] [-i IP] [--project PROJECT] [-s name=value]
                [-S project.settings]

HTTP API server for Scrapy project.

optional arguments:
  -h, --help            show this help message and exit
  -p PORT, --port PORT  port number to listen on
  -i IP, --ip IP        IP address the server will listen on
  --project PROJECT     project name from scrapy.cfg
  -s name=value, --set name=value
                        set/override setting (may be repeated)
  -S project.settings, --settings project.settings
                        custom project settings module path
```

CHAPTER 5

Configuration

You can pass custom settings to Scrapyrt using `-S` option (see *Command line arguments*):

```
scrapyrt -S config
```

Scrapyrt imports passed module, so it should be in one of the directories on `sys.path`.

Another way to configure server is to use `-s key=value` option:

```
scrapyrt -s TIMEOUT_LIMIT=120
```

Settings passed using `-s` option have the highest priority, settings passed in `-S` configuration module have priority higher than default settings.

5.1 Available settings

5.1.1 SERVICE_ROOT

Root server resource which is used to initialize Scrapyrt application. You can pass custom resource here and start Scrapyrt with it.

Default: `scrapyrt.resources.RealtimeApi`.

5.1.2 CRAWL_MANAGER

Crawl manager that is used to create and control crawl. You can override default crawl manager and pass path to custom class here.

Default: `scrapyrt.core.CrawlManager`.

5.1.3 RESOURCES

Dictionary where keys are resource URLs and values are resource classes. Used to setup Scrapyrt application with proper resources. If you want to add some additional resources - this is the place to add them.

Default:

```
RESOURCES = {
    'crawl.json': 'scrapyrt.resources.CrawlResource',
}
```

5.1.4 LOG_DIR

Path to directory to store crawl logs from running spiders.

Default: `log` directory.

5.1.5 TIMEOUT_LIMIT

Use this setting to limit crawl time.

Default: `1000`.

5.1.6 DEBUG

Run Scrapyrt in debug mode - in case of errors you will get Python tracebacks in response, for example:

```
{
    "status": "ok"
    "spider_name": "dmoz",
    "stats": {
        "start_time": "2014-12-29 17:26:11",
        "spider_exceptions/Exception": 1,
        "finish_time": "2014-12-29 17:26:11",
        "finish_reason": "finished",
        "downloader/response_status_count/200": 1,
        "downloader/response_count": 1,
        "downloader/response_bytes": 8494,
        "downloader/request_method_count/GET": 1,
        "downloader/request_count": 1,
        "downloader/request_bytes": 247,
        "log_count/DEBUG": 1,
        "log_count/ERROR": 1,
        "log_count/INFO": 4,
        "response_received_count": 1,
        "scheduler/dequeued": 1,
        "scheduler/dequeued/memory": 1,
        "scheduler/enqueued": 1,
        "scheduler/enqueued/memory": 1
    },
    "items": [],
    "items_dropped": [],
    "errors": [
        "Traceback (most recent call last): [...] \nexceptions.Exception: \n"
```

(continues on next page)

(continued from previous page)

```
    ],
}
```

Default: `True`.

5.1.7 PROJECT_SETTINGS

Automatically picked up from scrapy.cfg during initialization.

5.1.8 LOG_FILE

Path to file to store logs from Scrapyrt with daily rotation.

Default: `None`. Writing log to file is disabled by default.

5.1.9 LOG_ENCODING

Encoding that's used to encode log messages.

Default: `utf-8`.

5.2 Spider settings

Scrapyrt overrides some Scrapy project settings by default and most importantly it disables some Scrapy extensions:

```
"EXTENSIONS": {
    'scrapy.contrib.logstats.LogStats': None,
    'scrapy.webservice.WebService': None,
    'scrapy.telnet.TelnetConsole': None,
    'scrapy.contrib.throttle.AutoThrottle': None
}
```

There's usually no need and thus no simple way to change those settings, but if you have reason to do so you need to override `get_project_settings` method of `scrapyrt.core.CrawlManager`.

CHAPTER 6

Logging

ScrapyRT supports Scrapy logging with some limitations.

For each crawl it creates handler that's attached to the root logger and collects log records for which it can determine what spider object current log is related to. The only way to pass object to the log record is `extra` argument (see explanation and another usage example here):

```
logger.debug('Log message', extra={'spider': spider})
```

Spider object is passed by default in Spider.logger and Spider.log backwards compatibility wrapper so you don't have to pass it yourself if you're using them. All logs record that don't have reference to spider object or reference another spider object in the same process will be ignored.

Spider logging setup in ScrapyRT happens only after spider object instantiation, so logging from `Spider.__init__` method as well as logging during middleware, pipeline or extension instantiation is not supported due to limitations of initialization order in Scrapy.

Also ScrapyRT doesn't support LOG_STDOUT - if you're using `print` statements in a spider they will never be logged to any log file. Reason behind this is that there's no way to filter such log records and they will appear in all log files for crawls that are running simultaneously. This is considered harmful and is not supported. But if you still want to save all stdout to some file - you can create custom *SERVICE_ROOT* where you can setup logging stdout to file using approach described in Python Logging HOWTO or redirect stdout to a file using bash redirection syntax, supervisord logging etc.