pomp Documentation

Release 0.3.0

Evgeniy Tatarkin

May 03, 2017

Contents

Pomp is a screen scraping and web crawling framework. Pomp is inspired by and similar to Scrapy, but has a simpler implementation that lacks the hard Twisted dependency.

Features:

- Pure python
- Only one dependency for Python 2.x - concurrent.futures (backport of package for Python 2.x)
- Supports one file applications; Pomps doesn't force a specific project layout or other restrictions.
- Pomp is a meta framework like Paste: you may use it to create your own scraping framework.
- Extensible networking: you may use any sync or async method.
- No parsing libraries in the core; use you preferred approach.
- Pomp instances may be distributed and are designed to work with an external queue.

Pomp makes no attempt to accomodate:

- redirects
- proxies
- caching
- database integration
- cookies
- authentication
- etc.

If you want proxies, redirects, or similar, you may use the excellent requests library as the Pomp downloader.

Pomp examples

Pomp docs

Pomp is written and maintained by Evgeniy Tatarkin and is licensed under the BSD license.

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Pomp.

Quickstart

Pomp is fun to use, incredibly easy for basic applications.

A Minimal Application

For a minimal application all you need is to define you crawler by inherit *base.BaseCrawler*

```
import re
from pomp.core.base import BaseCrawler
from pomp.contrib.urllibtools import UrllibHttpRequest


python_sentence_re = re.compile('[\w\s]{0,}python[\s\w]{0,}', re.I | re.M)


class MyCrawler(BaseCrawler):
    """Extract all sentences with `python` word"""
    ENTRY_REQUESTS = UrllibHttpRequest('http://python.org/news')  # entry point

    def extract_items(self, response):
        for i in python_sentence_re.findall(response.body.decode('utf-8')):
            sentence = i.strip()
            print("Sentence: {}".format(sentence))
            yield sentence


if __name__ == '__main__':
    from pomp.core.engine import Pomp
```

```
    from pomp.contrib.urllibtools import UrllibDownloader

    pomp = Pomp(
        downloader=UrllibDownloader(),
    )

    pomp.pump(MyCrawler())
```

Item pipelines

For processing extracted items pomp has pipelines mechanism. Define pipe by subclass of *base.BasePipeline* and pass it to *engine.Pomp* constructor.

Pipe calls one by one

Example pipelines for filtering items with length less the 10 symbols and printing sentence:

```
class FilterPipeline(BasePipeline):
    def process(self, crawler, item):
        # None - skip item for following processing
        return None if len(item) < 10 else item

class PrintPipeline(BasePipeline):
    def process(self, crawler, item):
        print('Sentence:', item, ' length:', len(item))
        return item # return item for following processing

pomp = Pomp(
    downloader=UrllibDownloader(),
    pipelines=(FilterPipeline(), PrintPipeline(),)
)
```

See *Simple pipelines*

Custom downloader

For download data from source target application can define downloader to implement special protocols or strategies.

Custom downloader must be subclass of *base.BaseDownloader*

For example downloader fetching data by requests package.

```
import requests as requestslib
from pomp.core.base import BaseDownloader, BaseCrawlException
from pomp.core.base import BaseHttpRequest, BaseHttpResponse


class ReqRequest(BaseHttpRequest):
    def __init__(self, url):
        self.url = url


class ReqResponse(BaseHttpResponse):
    def __init__(self, request, response):
        self.request = request
```

```
        if not isinstance(response, Exception):
            self.body = response.text

    def get_request(self):
        return self.request


class RequestsDownloader(BaseDownloader):

    def process(self, crawler, request):
        try:
            return ReqResponse(request, requestslib.get(request.url))
        except Exception as e:
            print('Exception on %s: %s', request, e)
            return BaseCrawlException(request, exception=e)


if __name__ == '__main__':
    from pomp.core.base import BaseCrawler
    from pomp.core.engine import Pomp

    class Crawler(BaseCrawler):
        ENTRY_REQUESTS = ReqRequest('http://python.org/news/')

        def extract_items(self, response):
            print(response.body)

        def next_requests(self, response):
            return None  # one page crawler

    pomp = Pomp(
        downloader=RequestsDownloader(),
    )

    pomp.pump(Crawler())
```

Downloader middleware

For hook request before it executed by downloader or response before it passed to crawler in pomp exists middlewares framework.

Middleware must be subclass of *base.BaseMiddleware*.

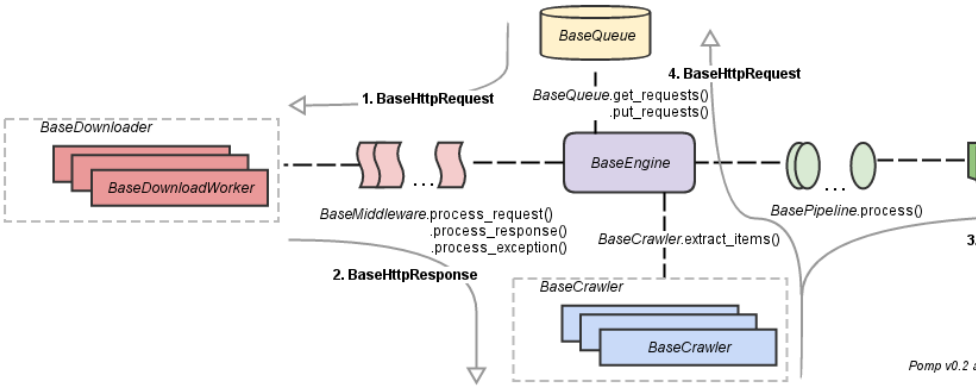
Each request will be passed to middlewares one by one in order it will passed to downloader. Each response/exception will be passed to middlewares one by one in reverse order.

For example statistic middleware:

```
from pomp.core.base import BaseMiddleware

class StatisticMiddleware(BaseMiddleware):
    def __init__(self):
        self.requests = self.responses = self.exceptions = 0

    def process_request(self, request, crawler, downloader):
        self.requests += 1
        return request
```

```
    def process_response(self, response, crawler, downloader):
        self.responses += 1
        return response

    def process_exception(self, exception, crawler, downloader):
        self.exceptions += 1
        return exception
```

Architecture



Similarly to Scrapy architecture

1. Request (*base.BaseHttpRequest*) from queue (*base.BaseQueue*) by engine (`base.BaseEngine`) passed to middlewares (*base.BaseMiddleware*) and then executed by downloader (*base.BaseDownloader*). Downloader can process request in concurrent way and return deferred result (*utils.Planned*).
2. Response (*base.BaseHttpResponse*) passed back to middlewares in reverse order and then to the crawler (*base.BaseCrawler*). Crawler can process response in concurrent way and return deferred result like downloader.
3. Extracted data passed to item pipeline.
4. Next requests if they exists will be putted to the queue.

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

API

This part of the documentation documents all the public classes and functions in pomp.

Contrib

Urllib

Downloader and middleware implementations.

- Downloaders: Fetches data by standard *urllib.urlopen* (Python 3.x) or *urllib2.urlopen* (Python 2.7+)

class pomp.contrib.urllibtools.**UrllibAdapterMiddleware**
Middlerware for adapting urllib.Request to *pomp.core.base.BaseHttpRequest*

class pomp.contrib.urllibtools.**UrllibDownloader**(*timeout=None*)
Simplest downloader

Parameters **timeout** – request timeout in seconds

class pomp.contrib.urllibtools.**UrllibHttpRequest**(*url*, *data=None*, *headers={}*, *origin_req_host=None*, *unverifiable=False*, *method=None*)
Adapter for urllib request to *pomp.core.base.BaseHttpRequest*

class pomp.contrib.urllibtools.**UrllibHttpResponse**(*request*, *response*)
Adapter for urllib response to *pomp.core.base.BaseHttpResponse*

Concurrent future

Concurrent downloaders

class pomp.contrib.concurrenttools.**ConcurrentCrawler**(*worker_class*, *worker_kwargs=None*, *pool_size=5*)

Concurrent ProcessPoolExecutor crawler

Parameters

- **pool_size** – pool size of ProcessPoolExecutor
- **timeout** – request timeout in seconds

class pomp.contrib.concurrenttools.**ConcurrentDownloader**(*worker_class*, *worker_kwargs=None*, *pool_size=5*)

Concurrent ProcessPoolExecutor downloader

Parameters

- **pool_size** – size of ThreadPoolExecutor
- **timeout** – request timeout in seconds

class pomp.contrib.concurrenttools.**ConcurrentUrllibDownloader**(*pool_size=5*, *timeout=None*)

Concurrent ProcessPoolExecutor downloader for fetching data with urllib pomp.contrib.SimpleDownloader

Parameters

- **pool_size** – pool size of ProcessPoolExecutor
- **timeout** – request timeout in seconds

Simple pipelines

Simple pipelines

class pomp.contrib.pipelines.**CsvPipeline**(*output_file*, **args*, ***kwargs*)

Save items to CSV format

Params **args* and ***kwargs* passed to csv.writer constuctor.

Parameters **output_file** – Filename of file-like object or a file object. If *output_file* is a file-like object, then the file will remain open after the pipe is stopped.

class pomp.contrib.pipelines.**UnicodeCsvWriter**(*f*, *dialect=<class 'csv.excel'>*, *encoding='utf-8'*, ***kwds*)

A CSV writer that writes rows to CSV file *f* with the given encoding.

Item and Field

class pomp.contrib.item.**Item**(**args*, ***kwargs*)

OrderedDict subclass

Engine

Engine

class pomp.core.engine.**Pomp**(*downloader*, *middlewares=None*, *pipelines=None*, *queue=None*, *breadth_first=False*)

Configuration object

This class glues together all parts of a Pomp instance:

•Downloader implementation and middleware

•Item pipelines

•Crawler

Parameters

- **downloader** – *pomp.core.base.BaseDownloader*
- **middlewares** – list of middlewares, instances of BaseMiddleware
- **pipelines** – list of item pipelines *pomp.core.base.BasePipeline*
- **queue** – external queue, instance of *pomp.core.base.BaseQueue*
- **breadth_first** – use BFO order or DFO order, sensibly if used internal queue only

LOCK_FACTORY()

allocate_lock() -> lock object (allocate() is an obsolete synonym)

Create a new lock object. See help(type(threading.Lock())) for information about locks.

pump(*crawler*)

Start crawling

Parameters **crawler** – instance of *pomp.core.base.BaseCrawler*

Interfaces

Base classes

Note: All classes in this package must be subclassed.

exception pomp.core.base.**BaseCrawlException**(*request=None*, *response=None*, *exception=None*, *exc_info=None*)

Download exception interface

Parameters

- **request** – request raises this exception
- **response** – response raises this exception
- **exception** – original exception
- **exc_info** – result of *sys.exc_info* call

class pomp.core.base.**BaseCrawler**

Crawler interface

The crawler must implement two tasks:

•Extract data from response

•Extract urls from response for follow-up processing

Each crawler must have one or more url starting points. To set the entry urls, declare them as class attribute ENTRY_REQUESTS:

```
class MyGoogleCrawler(BaseCrawler):
    ENTRY_REQUESTS = 'http://google.com/'
    ...
```

ENTRY_REQUESTS may be a list of urls or list of requests (instances of *BaseHttpRequest*).

extract_items(*response*)
Parse page and extract items.

May be *awaitable*.

Parameters **response** – the instance of *BaseHttpResponse*

Return type item/items of any type or type of *pomp.contrib.item.Item* or request/requests type of *BaseHttpRequest* or string/strings for following processing as requests

next_requests(*response*)
Returns follow-up requests for processing.

Called after the *extract_items* method.

May be *awaitable*.

Note Subclass may not implement this method. Next requests may be returned with items in *extrat_items* method.

Parameters **response** – the instance of *BaseHttpResponse*

Return type None or request or requests (instance of *BaseHttpRequest* or str). None response indicates that that this page does not any urls for follow-up processing.

on_processing_done(*response*)
Called when request/response was fully processed by middlewares, this crawler and and pipelines.

May be awaitable.

Parameters **response** – the instance of *BaseHttpResponse*

class pomp.core.base.**BaseDownloadWorker**
Download worker interface

process(*request*)
Execute request

May be awaitable.

Parameters **request** – instance of *BaseHttpRequest*

Return type instance of *BaseHttpResponse* or *BaseCrawlException* or *Planned* or *asyncio.Future* for async behavior

class pomp.core.base.**BaseDownloader**
Downloader interface

The downloader must implement one task:

- make http request and fetch response.

get_workers_count()

Return type count of workers (pool size), by default 0

process(*crawler*, *request*)
Execute request

May be *awaitable*.

Parameters

- **crawler** – crawler that extracts items
- **request** – instances of *BaseHttpRequest*

Return type instance of *BaseHttpResponse* or *BaseCrawlException* or *Planned* or *asyncio.Future* object for async behavior

start(*crawler*)
Prepare downloader before processing starts.

May be *awaitable*.

Parameters **crawler** – crawler that extracts items

stop(*crawler*)
Stop downloader.

May be *awaitable*.

Parameters **crawler** – crawler that extracts items

class pomp.core.base.**BaseHttpRequest**
HTTP request interface

class pomp.core.base.**BaseHttpResponse**
HTTP response interface

get_request()
Request *BaseHttpRequest*

class pomp.core.base.**BaseMiddleware**
Middleware interface

process_exception(*exception*, *crawler*, *downloader*)
Handle exception

May be *awaitable*.

Parameters

- **exception** – instance of *BaseCrawlException*
- **crawler** – instance of *BaseCrawler*
- **downloader** – instance of *BaseDownloader*

Return type changed response or `None` to skip processing of this exception

process_request(*request*, *crawler*, *downloader*)
Change request before it will be executed by downloader

May be *awaitable*.

Parameters

- **request** – instance of *BaseHttpRequest*
- **crawler** – instance of *BaseCrawler*
- **downloader** – instance of *BaseDownloader*

Return type changed request or `None` to skip execution of this request

process_response(*response*, *crawler*, *downloader*)
Modify response before content is extracted by the crawler.

May be *awaitable*.

Parameters

- **response** – instance of *BaseHttpResponse*
- **crawler** – instance of *BaseCrawler*
- **downloader** – instance of *BaseDownloader*

Return type changed response or `None` to skip processing of this response

class pomp.core.base.**BasePipeline**
Pipeline interface

The function of pipes are to:

•filter items

•change items

•store items

process(*crawler*, *item*)
Process extracted item

May be *awaitable*.

Parameters

- **crawler** – crawler that extracts items
- **item** – extracted item

Return type item or `None` if this item is to be skipped

start(*crawler*)
Initialize pipe

Open files and database connections, etc.

May be *awaitable*.

Parameters **crawler** – crawler that extracts items

stop(*crawler*)
Finalize pipe

Close files and database connections, etc.

May be *awaitable*.

Parameters **crawler** – crawler that extracts items

class pomp.core.base.**BaseQueue**
Blocking queue interface

get_requests(*count=None*)
Get from queue

Note: must block execution until item is available

Parameters **count** – count of requests to be processed by downloader in concurrent mode, None - downloader have not concurrency (workers). This param can be ignored.

Return type instance of *BaseRequest* or *Planned* or list of them

put_requests(*requests*)
Put to queue

Parameters **requests** – instance of *BaseRequest* or list of them

class pomp.core.base.**BaseRequest**
Request interface

class pomp.core.base.**BaseResponse**
Response interface

Utils

exception pomp.core.utils.**CancelledError**
The Planned was cancelled.

exception pomp.core.utils.**Error**
Base class for all planned-related exceptions.

exception pomp.core.utils.**NotDoneYetError**
The Planned was not completed.

class pomp.core.utils.**Planned**
Clone of Future object, but without thread conditions (locks).

Represents the result of an asynchronous computation.

add_done_callback(*fn*)
Attaches a callable that will be called when the future finishes.

Args:

fn: A callable that will be called with this future as its only argument when the future completes or is cancelled. If the future has already completed or been cancelled then the callable will be called immediately. These callables are called in the order that they were added.

cancel()
Cancel the future if possible.

Returns True if the future was cancelled, False otherwise. A future cannot be cancelled if it is running or has already completed.

cancelled()
Return True if the future was cancelled.

done()
Return True of the future was cancelled or finished executing.

result()
Return the result of the call that the future represents.

Returns: The result of the call that the future represents.

Raises: CancelledError: If the future was cancelled. Exception: If the call raised then that exception will be raised.

set_result(*result*)
Sets the return value of work associated with the future.

Should only be used by Executor implementations and unit tests.

Python Module Index

p

Index

R

S

U