Scrapy
beyond the first steps

Eugenio Lacuesta

Python Brasil, october 2018

scrapinghub

Hello everyone! I'm Eugenio

About me:

- Python developer from Montevideo, Uruguay
- Scrapinghubber since 2015
- Scrapy contributor

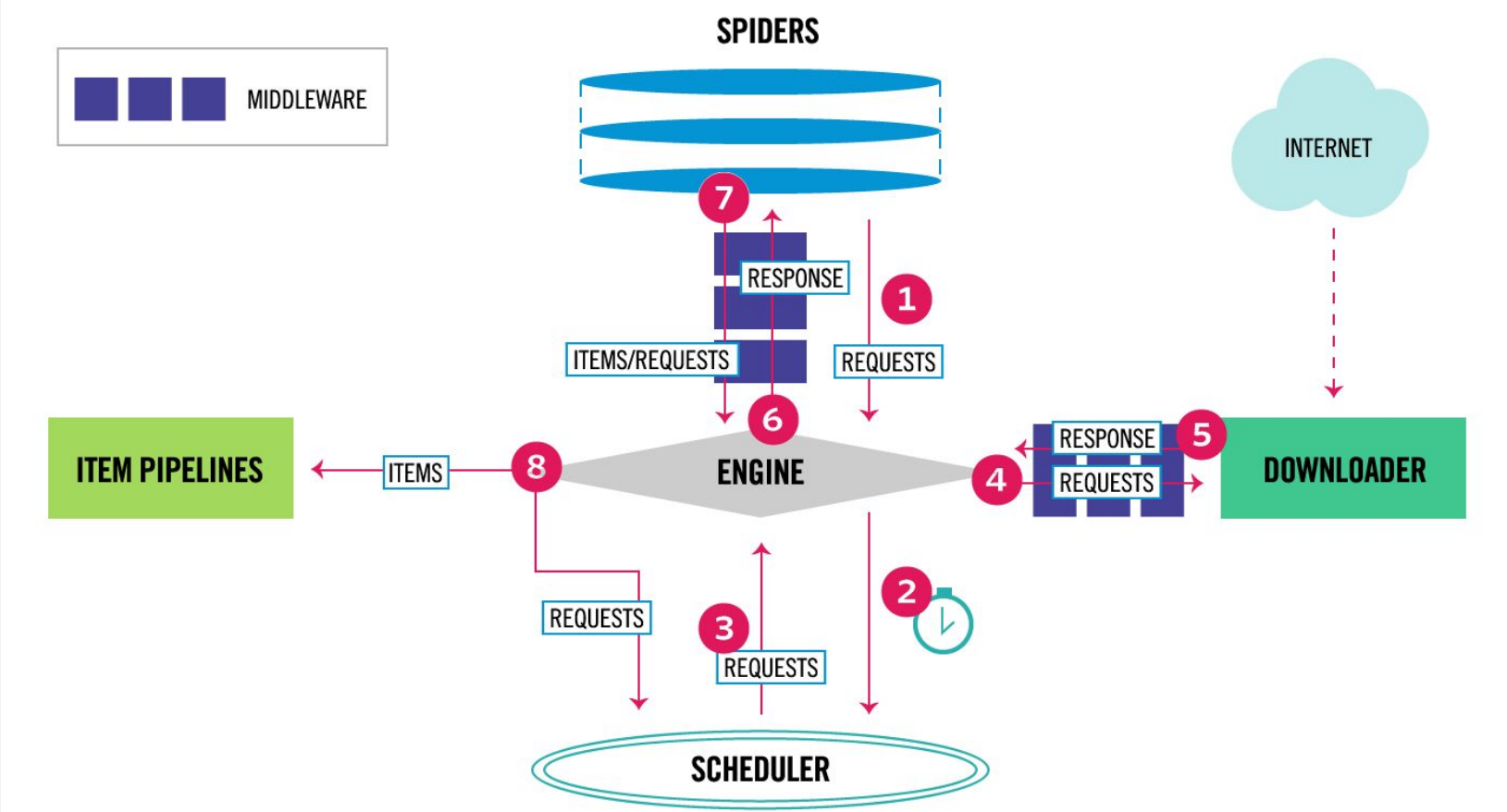
About this talk:

- Only basic Scrapy knowledge needed (spiders, requests, responses)
- Scrapy project and slides at **@elacuesta**

Talk summary

- Quick architecture overview
- Basic spider
- Item exporters
- Item pipeline
- Signals
- from_crawler factory method
- Downloader middleware

Architecture (https://doc.scrapy.org/en/latest/topics/architecture.html)



Basic spider - http://books.toscrape.com

```
from scrapy import Spider

class BooksSpider(Spider):
    name = 'books'
    start_urls = ['http://books.toscrape.com']

    def parse(self, response):
        for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
            yield response.follow(book_link, callback=self.parse_book)

    def parse_book(self, response):
        return {
            'url': response.url,
            'title': response.css('h1::text').get(),
            'price': float(response.css('p.price_color::text').re_first(r'(\d+.?\d*)')),
        }
```

Basic spider - http://books.toscrape.com

```
from scrapy import Spider

class BooksSpider(Spider):
    name = 'books'
    start_urls = ['http://books.toscrape.com']

    def parse(self, response):
        for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
            yield response.follow(book_link, callback=self.parse_book)

    def parse_book(self, response):
        return {
            'url': response.url,
            'title': response.css('h1::text').get(),
            'price': float(response.css('p.price_color::text').re_first(r'(\d+.?\d*)')),
        }
```

Basic spider - http://books.toscrape.com

```
from scrapy import Spider

class BooksSpider(Spider):
    name = 'books'
    start_urls = ['http://books.toscrape.com']

    def parse(self, response):
        for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
            yield response.follow(book_link, callback=self.parse_book)

    def parse_book(self, response):
        return {
            'url': response.url,
            'title': response.css('h1::text').get(),
            'price': float(response.css('p.price_color::text').re_first(r'(\d+.?\d*)')),
        }
```

Iterate over the links and produce requests to the specific books

Basic spider - http://books.toscrape.com

```
from scrapy import Spider

class BooksSpider(Spider):
    name = 'books'
    start_urls = ['http://books.toscrape.com']

    def parse(self, response):
        for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
            yield response.follow(book_link, callback=self.parse_book)

    def parse_book(self, response):
        return {
            'url': response.url,
            'title': response.css('h1::text').get(),
            'price': float(response.css('p.price_color::text').re_first(r'(\d+.?\d*)')),
        }
```

Extract information from each book page

```
$ scrapy crawl books -o books.json
2018-10-08 14:36:31 [scrapy.utils.log] INFO: Scrapy 1.5.0 started (bot: pybr2018)
(...)
2018-10-08 14:36:34 [scrapy.core.engine] INFO: Spider closed (finished)

$ cat books.json | jq .
[
  {
    "url": "http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html",
    "title": "A Light in the Attic",
    "price": 51.77,
  },
  {
    "url":
"http://books.toscrape.com/catalogue/scott-pilgrims-precious-little-life-scott-pilgrim-1_987/index.html",
    "title": "Scott Pilgrim's Precious Little Life (Scott Pilgrim #1)",
    "price": 52.29,
  },
...

$
```

Item exporters

scrapinghub

Item exporters

In the previous example, the `-o books.json` part indicated Scrapy that we wanted the output in JSON format.

Scrapy comes with a few built-in exporters (JSON, XML, CSV)

Custom item exporters are supported though the `FEED_EXPORTERS` setting:

```
FEED_EXPORTERS = {
    'yaml': 'pybr2018.exporters.YAMLItemExporter',
}
```

Project's settings.py file

YAML Item exporter

```
from ruamel.yaml import YAML
from scrapy.exporters import BaseItemExporter


class YAMLItemExporter(BaseItemExporter):

    def __init__(self, file, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.file = file
        self.yaml = YAML()
        self.yaml.encoding = self.encoding

    def export_item(self, item):
        self.yaml.dump([dict(item)], self.file)
```

YAML Item exporter

```
from ruamel.yaml import YAML
from scrapy.exporters import BaseItemExporter


class YAMLItemExporter(BaseItemExporter):

    def __init__(self, file, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.file = file
        self.yaml = YAML()
        self.yaml.encoding = self.encoding

    def export_item(self, item):
        self.yaml.dump([dict(item)], self.file)
```

Inherit from Scrapy's Base Item Exporter

YAML Item exporter

```
from ruamel.yaml import YAML
from scrapy.exporters import BaseItemExporter


class YAMLItemExporter(BaseItemExporter):

    def __init__(self, file, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.file = file
        self.yaml = YAML()
        self.yaml.encoding = self.encoding

    def export_item(self, item):
        self.yaml.dump([dict(item)], self.file)
```

Initialize the base exporter.
Store the output file pointer and
create the YAML exporter

YAML Item exporter

```
from ruamel.yaml import YAML
from scrapy.exporters import BaseItemExporter


class YAMLItemExporter(BaseItemExporter):

    def __init__(self, file, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.file = file
        self.yaml = YAML()
        self.yaml.encoding = self.encoding

    def export_item(self, item):
        self.yaml.dump([dict(item)], self.file)
```

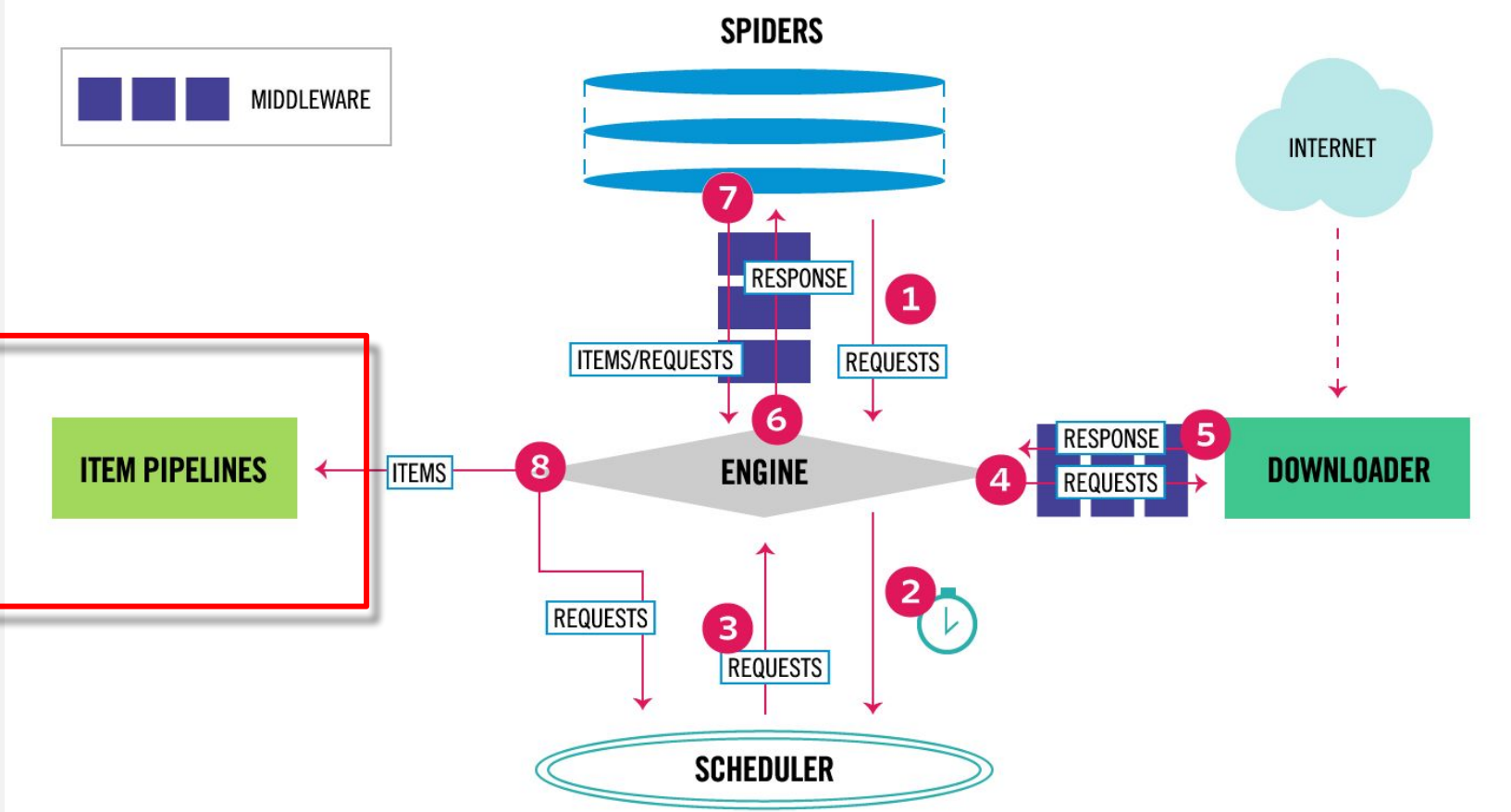
Serialize each item

```
$ scrapy crawl books -o books.yaml
2018-10-08 15:43:01 [scrapy.utils.log] INFO: Scrapy 1.5.0 started (bot: pybr2018)
(...several log lines...)
2018-10-08 15:43:04 [scrapy.core.engine] INFO: Spider closed (finished)

$ head -n 12 books.yaml
- url: http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html
  title: A Light in the Attic
  price: 51.77
- url: http://books.toscrape.com/catalogue/rip-it-up-and-start-again_986/index.html
  title: Rip it Up and Start Again
  price: 35.02
- url: http://books.toscrape.com/catalogue/set-me-free_988/index.html
  title: Set Me Free
  price: 17.46

$
```

Item pipeline

scrapinghub

Item pipeline



Item pipeline

After an item is scraped, it is sent through the Item Pipeline.

Each Pipeline stage is just a Python class that implements a `process_item` method, which receives the item and the spider object that produced it

This method can perform any action on the item, and it can also decide if the item should be sent to the next stage (by `return`ing it) or discarded (by raising the `DropItem` exception)

Item pipeline - Validate books

```
class BooksSpider(Spider):
      custom_settings = {
            'ITEM_PIPELINES': {'pybr2018.pipelines.ValidateBookPipeline': 100}
      }
```

```
from scrapy.exceptions import DropItem
import jsonschema

class ValidateBookPipeline:
      schema = {...}  # a valid JSON schema

      def process_item(self, item, spider):
            try:
                  jsonschema.validate(dict(item), self.schema)
            except jsonschema.ValidationError as ex:
                  raise DropItem(ex.message)
            else:
                  return item
```

Item pipeline - Validate books

```
class BooksSpider(Spider):
      custom_settings = {
            'ITEM_PIPELINES': {'pybr2018.pipelines.ValidateBookPipeline': 100}
      }
```

Enable the Pipeline in the spider class

```
from scrapy.exceptions import DropItem
import jsonschema

class ValidateBookPipeline:
      schema = {...}  # a valid JSON schema

      def process_item(self, item, spider):
            try:
                  jsonschema.validate(dict(item), self.schema)
            except jsonschema.ValidationError as ex:
                  raise DropItem(ex.message)
            else:
                  return item
```

Item pipeline - Validate books

```
class BooksSpider(Spider):
      custom_settings = {
            'ITEM_PIPELINES': {'pybr2018.pipelines.ValidateBookPipeline': 100}
      }
```

```
from scrapy.exceptions import DropItem
import jsonschema

class ValidateBookPipeline:
      schema = {...}  # a valid JSON schema

      def process_item(self, item, spider):
            try:
                  jsonschema.validate(dict(item), self.schema)
            except jsonschema.ValidationError as ex:
                  raise DropItem(ex.message)
            else:
                  return item
```

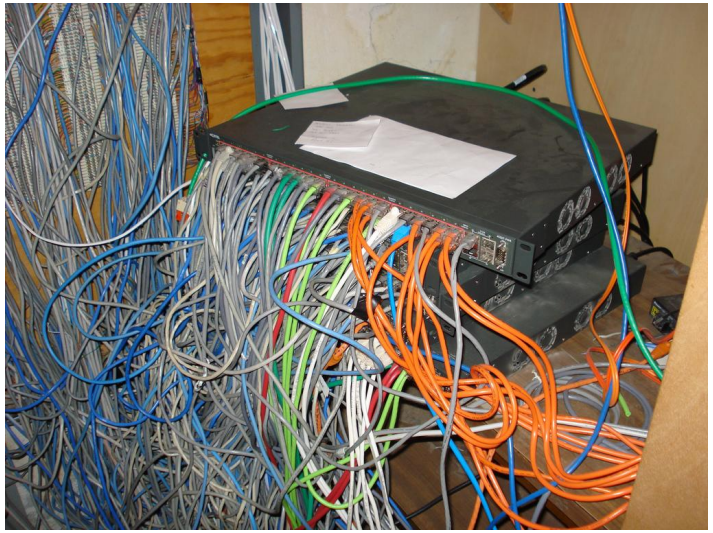
An actual JSON schema (missing here because it's too long)

If the validation fails raise a DropItem exception to indicate Scrapy the item should be discarded

```
$ scrapy crawl books
2018-10-08 17:18:06 [scrapy.utils.log] INFO: Scrapy 1.5.0 started (bot: pybr2018)
(...)
2018-10-08 17:18:08 [scrapy.core.scraper] WARNING: Dropped: 52.15 is greater than the maximum of
50
{'url': 'http://books.toscrape.com/catalogue/the-black-maria_991/index.html', 'title': 'The Black
Maria', 'price': 52.15}
2018-10-08 17:18:08 [scrapy.core.scraper] WARNING: Dropped: 54.23 is greater than the maximum of
50
{'url': 'http://books.toscrape.com/catalogue/sapiens-a-brief-history-of-humankind_996/index.html',
'title': 'Sapiens: A Brief History of Humankind', 'price': 54.23}
2018-10-08 17:18:08 [scrapy.core.scraper] WARNING: Dropped: 'The Coming Woman: A Novel Based on
the Life of the Infamous Feminist, Victoria Woodhull' is too long
{'url':
'http://books.toscrape.com/catalogue/the-coming-woman-a-novel-based-on-the-life-of-the-infamous-fe
minist-victoria-woodhull_993/index.html', 'title': 'The Coming Woman: A Novel Based on the Life of
the Infamous Feminist, Victoria Woodhull', 'price': 17.93}
(...)
2018-10-08 17:18:09 [scrapy.core.engine] INFO: Spider closed (finished)

$
```

Signals

Problem - Stateful server



https://www.flickr.com/photos/dcmorton/2446443463/

Now imagine the site stores information about the followed links, and returns responses based on the user's navigation patterns.
Scheduling concurrent requests is not an option anymore.

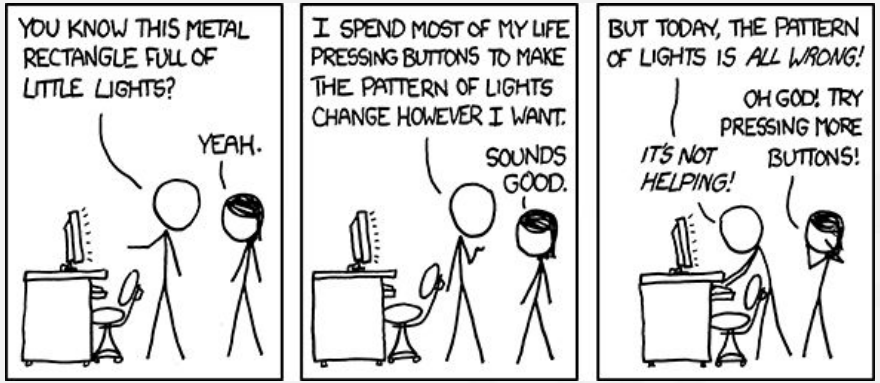
Possible solutions:

- Sequential crawl
- Parallel sessions

(One) solution - Sequential crawling

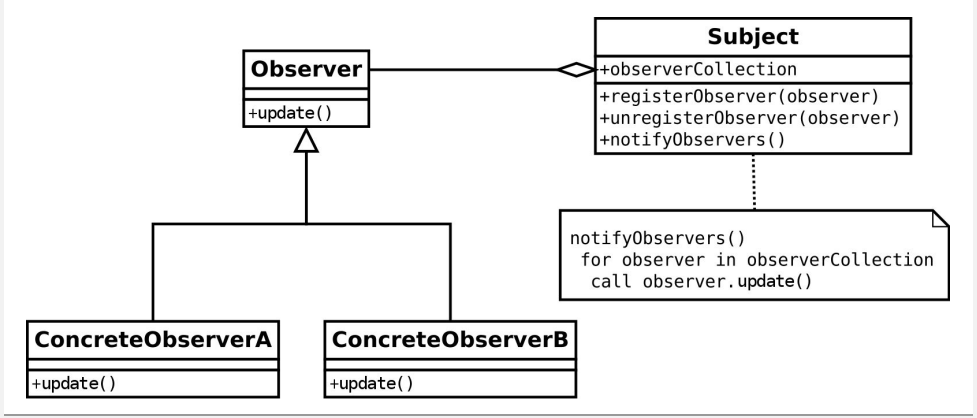
We'll take advantage of two design patterns employed on Scrapy:

- Observer pattern (signals)
- Factory pattern (from_crawler class method)



https://www.xkcd.com/722/

Signals



https://commons.wikimedia.org/wiki/File:Observer_w_update.svg

Observer design pattern: attach handlers to act on system events.

Several available events to handle:

- Engine is started
- Response is downloaded
- Item is scraped
- Spider is idle
- etc

`spider_idle` signal

From the Documentation:

Fired when the spider has no further requests scheduled/waiting to be downloaded or items being processed by the Item Pipeline

Provides *one* approach to sequential crawling

`from_crawler` class method

scrapinghub

`from_crawler` class method

The main entry point to Scrapy API is the *Crawler* object, passed (...) through the `from_crawler` class method. This object provides access to all Scrapy core components, and it's the only way (...) to access them and hook their functionality into Scrapy.

```
class MySpider(scrapy.Spider):
      @classmethod
      def from_crawler(cls, crawler, *args, **kwargs):
            spider = super().from_crawler(crawler, *args, **kwargs)
            crawler.signals.connect(spider.schedule_request, signal=scrapy.signals.spider_idle)
            return spider
```

`from_crawler` class method

The main entry point to Scrapy API is the *Crawler* object, passed (...) through the `from_crawler` class method. This object provides access to all Scrapy core components, and it's the only way (...) to access them and hook their functionality into Scrapy.

```python
class MySpider(scrapy.Spider):
    @classmethod
    def from_crawler(cls, crawler, *args, **kwargs):
        spider = super().from_crawler(crawler, *args, **kwargs)
        crawler.signals.connect(spider.schedule_request, signal=scrapy.signals.spider_idle)
        return spider
```

Factory design pattern

Connect a handler to the signal

Putting it all together

```
class SequentialBooksSpider(BooksSpider):
	name = 'books-sequential'
	pending = collections.deque()

	@classmethod
	def from_crawler(cls, crawler, *args, **kwargs):
		spider = super().from_crawler(crawler, *args, **kwargs)
		crawler.signals.connect(spider.schedule_request, signal=scrapy.signals.spider_idle)
		return spider

	def schedule_request(self):
		if self.pending:
			request = self.pending.popleft()
			self.crawler.engine.crawl(request, self)

	def parse(self, response):
		for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
			self.pending.append(response.follow(book_link, callback=self.parse_book))
			self.pending.append(response.request.replace(dont_filter=True, callback=self.parse_dummy))

	def parse_dummy(self, response):
		self.logger.info('Back at the main page')
```

Putting it all together

```
class SequentialBooksSpider(BooksSpider):
	name = 'books-sequential'
	pending = collections.deque()

	@classmethod
	def from_crawler(cls, crawler, *args, **kwargs):
		spider = super().from_crawler(crawler, *args, **kwargs)
		crawler.signals.connect(spider.schedule_request, signal=scrapy.signals.spider_idle)
		return spider

	def schedule_request(self):
		if self.pending:
			request = self.pending.popleft()
			self.crawler.engine.crawl(request, self)

	def parse(self, response):
		for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
			self.pending.append(response.follow(book_link, callback=self.parse_book))
			self.pending.append(response.request.replace(dont_filter=True, callback=self.parse_dummy))

	def parse_dummy(self, response):
		self.logger.info('Back at the main page')
```

Inherit from our previous spider to reuse the parse_book method

Container to store pending requests

Putting it all together

```
class SequentialBooksSpider(BooksSpider):
    name = 'books-sequential'
    pending = collections.deque()

    @classmethod
    def from_crawler(cls, crawler, *args, **kwargs):
        spider = super().from_crawler(crawler, *args, **kwargs)
        crawler.signals.connect(spider.schedule_request, signal=scrapy.signals.spider_idle)
        return spider

    def schedule_request(self):
        if self.pending:
            request = self.pending.popleft()
            self.crawler.engine.crawl(request, self)

    def parse(self, response):
        for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
            self.pending.append(response.follow(book_link, callback=self.parse_book))
            self.pending.append(response.request.replace(dont_filter=True, callback=self.parse_dummy))

    def parse_dummy(self, response):
        self.logger.info('Back at the main page')
```

Factory method:
attach signal handler

Signal handler: crawl next
request when the spider is idle

Putting it all together

```
class SequentialBooksSpider(BooksSpider):
	name = 'books-sequential'
	pending = collections.deque()

	@classmethod
	def from_crawler(cls, crawler, *args, **kwargs):
		spider = super().from_crawler(crawler, *args, **kwargs)
		crawler.signals.connect(spider.schedule_request, signal=scrapy.signals.spider_idle)
		return spider

	def schedule_request(self):
		if self.pending:
			request = self.pending.popleft()
			self.crawler.engine.crawl(request, self)

	def parse(self, response):
		for book_link in response.css('article.product_pod h3 a::attr(href)').getall():
			self.pending.append(response.follow(book_link, callback=self.parse_book))
			self.pending.append(response.request.replace(dont_filter=True, callback=self.parse_dummy))

	def parse_dummy(self, response):
		self.logger.info('Back at the main page')
```

Store book requests instead of scheduling directly

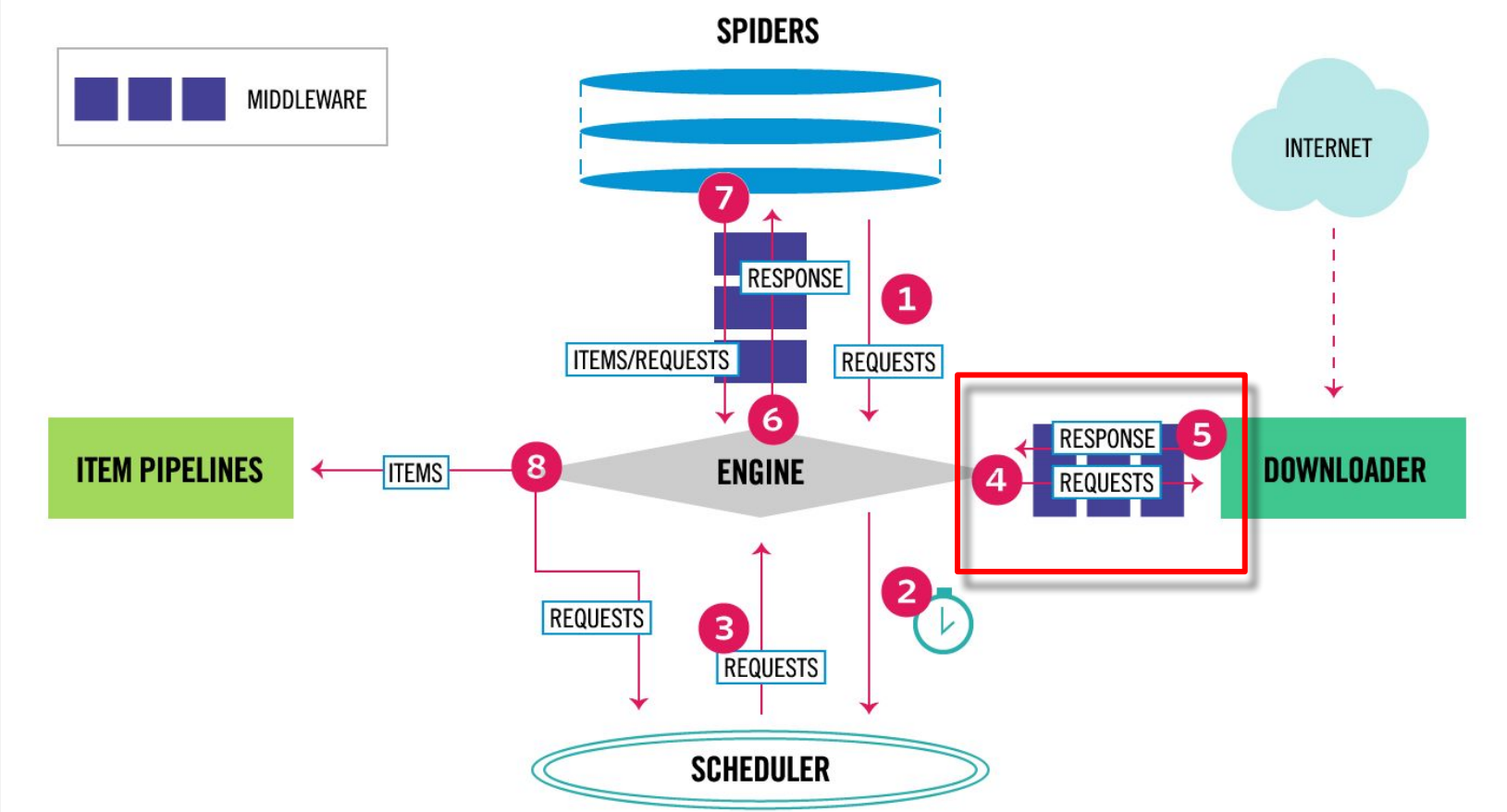
Simulate going back to the main page

```
$ scrapy crawl books-sequential
2018-10-09 12:54:19 [scrapy.utils.log] INFO: Scrapy 1.5.0 started (bot: pybr2018)
(...)
2018-10-09 12:54:20 [books-sequential] INFO: Scheduling:
http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html
2018-10-09 12:54:21 [books-sequential] INFO: Extracting:
http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html
2018-10-09 12:54:21 [books-sequential] INFO: Scheduling: http://books.toscrape.com
2018-10-09 12:54:21 [books-sequential] INFO: Back at the main page
2018-10-09 12:54:21 [books-sequential] INFO: Scheduling:
http://books.toscrape.com/catalogue/tipping-the-velvet_999/index.html
2018-10-09 12:54:22 [books-sequential] INFO: Extracting:
http://books.toscrape.com/catalogue/tipping-the-velvet_999/index.html
2018-10-09 12:54:22 [books-sequential] INFO: Scheduling: http://books.toscrape.com
2018-10-09 12:54:22 [books-sequential] INFO: Back at the main page
(...)
2018-10-09 12:54:30 [scrapy.core.engine] INFO: Spider closed (finished)

$
```

Downloader middleware

scrapinghub

Downloader middleware



Downloader middleware

The downloader middleware is a framework of hooks into Scrapy's request/response processing. It's a light, low-level system for globally altering Scrapy's requests and responses.

From the docs

More at https://doc.scrapy.org/en/latest/topics/downloader-middleware.html

Downloader middleware - SOAP example

In the following example we will use Zeep (https://python-zeep.readthedocs.io) to communicate with a SOAP web service that converts between Celsius and Fahrenheit.

The SOAP-related heavy lifting will be handled by a downloader middleware, making the process relatively transparent to the spider.

```python
class TemperatureConversionMiddleware:

    def __init__(self):
        self.client = zeep.Client('https://www.w3schools.com/xml/tempconvert.asmx?WSDL')

    def process_request(self, request, spider):
        # ...
        body = self.client.create_message(request, ...)
        return Request(
            url=request.url,
            method='POST',
            body=lxml.etree.tostring(body),
            # ...
        )

    def process_response(self, request, response, spider):
        # ...
        request.meta['result'] = process_reply(response, ...)
        return response
```

```
class TemperatureConversionMiddleware:

    def __init__(self):
        self.client = zeep.Client('https://www.w3schools.com/xml/tempconvert.asmx?WSDL')

    def process_request(self, request, spider):
        # ...
        body = self.client.create_message(request, ...)
        return Request(
            url=request.url,
            method='POST',
            body=lxml.etree.tostring(body),
            # ...
        )

    def process_response(self, request, response, spider):
        # ...
        request.meta['result'] = process_reply(response, ...)
        return response
```

The actual middleware is longer, lots of stuff omitted in this slide!

```python
class TemperatureConversionMiddleware:

    def __init__(self):
        self.client = zeep.Client('https://www.w3schools.com/xml/tempconvert.asmx?WSDL')

    def process_request(self, request, spider):
        # ...
        body = self.client.create_message(request, ...)
        return Request(
            url=request.url,
            method='POST',
            body=lxml.etree.tostring(body),
            # ...
        )

    def process_response(self, request, response, spider):
        # ...
        request.meta['result'] = process_reply(response, ...)
        return response
```

```
class TemperatureConversionMiddleware:

    def __init__(self):
        self.client = zeep.Client('https://www.w3schools.com/xml/tempconvert.asmx?WSDL')

    def process_request(self, request, spider):
        # ...
        body = self.client.create_message(request, ...)
        return Request(
            url=request.url,
            method='POST',
            body=lxml.etree.tostring(body),
            # ...
        )

    def process_response(self, request, response, spider):
        # ...
        request.meta['result'] = process_reply(response, ...)
        return response
```

Process requests to the SOAP service. Use Zeep to create XML request bodies for Scrapy to send

```
class TemperatureConversionMiddleware:

    def __init__(self):
        self.client = zeep.Client('https://www.w3schools.com/xml/tempconvert.asmx?WSDL')

    def process_request(self, request, spider):
        # ...
        body = self.client.create_message(request, ...)
        return Request(
            url=request.url,
            method='POST',
            body=lxml.etree.tostring(body),
            # ...
        )

    def process_response(self, request, response, spider):
        # ...
        request.meta['result'] = process_reply(response, ...)
        return response
```

Process responses coming back from the SOAP service.
Use Zeep to parse the XML bodies

Spider - Temperature converter

```
class TemperatureSpider(scrapy.Spider):
      name = 'temperature'
      url = 'https://www.w3schools.com/xml/tempconvert.asmx'
      custom_settings = {
            'DOWNLOADER_MIDDLEWARES': {
                  'pybr2018.middlewares.temperature.TemperatureConversionMiddleware': 543,
            }
      }

      def start_requests(self):
            for operation in ('CelsiusToFahrenheit', 'FahrenheitToCelsius'):
                  for _ in range(5):
                        meta = {'operation_name': operation, 'source_value': random.uniform(0, 50)}
                        yield scrapy.Request(self.url, dont_filter=True, meta=meta)

      def parse(self, response):
            source_unit, destination_unit = response.meta['operation_name'].split('To')
            return {
                  'source': '{} {}'.format(response.meta['source_value'], source_unit),
                  'destination': '{} {}'.format(response.meta['result'], destination_unit),
            }
```

Spider - Temperature converter

```
class TemperatureSpider(scrapy.Spider):
    name = 'temperature'
    url = 'https://www.w3schools.com/xml/tempconvert.asmx'
    custom_settings = {
        'DOWNLOADER_MIDDLEWARES': {
            'pybr2018.middlewares.temperature.TemperatureConversionMiddleware': 543,
        }
    }

    def start_requests(self):
        for operation in ('CelsiusToFahrenheit', 'FahrenheitToCelsius'):
            for _ in range(5):
                meta = {'operation_name': operation, 'source_value': random.uniform(0, 50)}
                yield scrapy.Request(self.url, dont_filter=True, meta=meta)

    def parse(self, response):
        source_unit, destination_unit = response.meta['operation_name'].split('To')
        return {
            'source': '{} {}'.format(response.meta['source_value'], source_unit),
            'destination': '{} {}'.format(response.meta['result'], destination_unit),
        }
```

SOAP service base URL

Enable the downloader middleware

Spider - Temperature converter

```
class TemperatureSpider(scrapy.Spider):
    name = 'temperature'
    url = 'https://www.w3schools.com/xml/tempconvert.asmx'
    custom_settings = {
        'DOWNLOADER_MIDDLEWARES': {
            'pybr2018.middlewares.temperature.TemperatureConversionMiddleware': 543,
        }
    }

    def start_requests(self):
        for operation in ('CelsiusToFahrenheit', 'FahrenheitToCelsius'):
            for _ in range(5):
                meta = {'operation_name': operation, 'source_value': random.uniform(0, 50)}
                yield scrapy.Request(self.url, dont_filter=True, meta=meta)

    def parse(self, response):
        source_unit, destination_unit = response.meta['operation_name'].split('To')
        return {
            'source': '{} {}'.format(response.meta['source_value'], source_unit),
            'destination': '{} {}'.format(response.meta['result'], destination_unit),
        }
```

Produce requests with random temperature values for the SOAP service to convert

Spider - Temperature converter

```
class TemperatureSpider(scrapy.Spider):
    name = 'temperature'
    url = 'https://www.w3schools.com/xml/tempconvert.asmx'
    custom_settings = {
        'DOWNLOADER_MIDDLEWARES': {
            'pybr2018.middlewares.temperature.TemperatureConversionMiddleware': 543,
        }
    }

    def start_requests(self):
        for operation in ('CelsiusToFahrenheit', 'FahrenheitToCelsius'):
            for _ in range(5):
                meta = {'operation_name': operation, 'source_value': random.uniform(0, 50)}
                yield scrapy.Request(self.url, dont_filter=True, meta=meta)

    def parse(self, response):
        source_unit, destination_unit = response.meta['operation_name'].split('To')
        return {
            'source': '{} {}'.format(response.meta['source_value'], source_unit),
            'destination': '{} {}'.format(response.meta['result'], destination_unit),
        }
```

Produce items with the converted value, processed by the middleware

```
$ scrapy crawl temperature -o temperature.json
2018-10-10 10:38:41 [scrapy.utils.log] INFO: Scrapy 1.5.0 started (bot: pybr2018)
(...)
2018-10-10 10:38:42 [TemperatureConversionMiddleware] INFO: Creating request for "CelsiusToFahrenheit" operation
(...)
2018-10-10 10:38:43 [TemperatureConversionMiddleware] INFO: Processing response for "CelsiusToFahrenheit" operation
(...)
2018-10-10 10:38:43 [scrapy.core.engine] INFO: Spider closed (finished)

$ cat temperature.json | jq .
[
  {
    "source": "44.562162995713486 Celsius",
    "destination": "112.211893392284 Fahrenheit"
  },
  {
    "source": "4.497578874865576 Celsius",
    "destination": "40.095641974758 Fahrenheit"
  },
(...)

$
```

Scrapinghub is hiring!

We're a globally distributed team of over 140 scrapinghubbers who are passionate about scraping, web crawling, and data science

Come join the team: https://scrapinghub.com/jobs

scrapinghub

Questions?

scrapinghub

Obrigado! Gracias! Thanks!

Eugenio Lacuesta
lacuesta@scrapinghub.com

scrapinghub