Spiders

Spiders are classes which define how a certain site (or a group of sites) will be scraped, including how to perform the crawl (i.e. follow links) and how to extract structured data from their pages (i.e. scraping items). In other words, Spiders are the place where you define the custom behaviour for crawling and parsing pages for a particular site (or, in some cases, a group of sites).

For spiders, the scraping cycle goes through something like this:

1. You start by generating the initial Requests to crawl the first URLs, and specify a callback function to be called with the response downloaded from those requests.

   The first requests to perform are obtained by calling the :meth:`~scrapy.Spider.start_requests` method which (by default) generates :class:`~scrapy.Request` for the URLs specified in the :attr:`~scrapy.Spider.start_urls` and the :attr:`~scrapy.Spider.parse` method as callback function for the Requests.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 19); *backlink*

   Unknown interpreted text role "meth".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 19); *backlink*

   Unknown interpreted text role "class".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 19); *backlink*

   Unknown interpreted text role "attr".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 19); *backlink*

   Unknown interpreted text role "attr".

2. In the callback function, you parse the response (web page) and return :ref:`item objects <topics-items>`, :class:`~scrapy.Request` objects, or an iterable of these objects. Those Requests will also contain a callback (maybe the same) and will then be downloaded by Scrapy and then their response handled by the specified callback.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 26); *backlink*

   Unknown interpreted text role "ref".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 26); *backlink*

   Unknown interpreted text role "class".

3. In callback functions, you parse the page contents, typically using :ref:`topics-selectors` (but you can also use BeautifulSoup, lxml or whatever mechanism you prefer) and generate items with the parsed data.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 33); *backlink*

   Unknown interpreted text role "ref".

4. Finally, the items returned from the spider will be typically persisted to a database (in some :ref:`Item Pipeline <topics-item-pipeline>`) or written to a file using :ref:`topics-feed-exports`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 37); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 37); *backlink*

Unknown interpreted text role "ref".

Even though this cycle applies (more or less) to any kind of spider, there are different kinds of default spiders bundled into Scrapy for different purposes. We will talk about those types here.

scrapy.Spider

This is the simplest spider, and the one from which every other spider must inherit (including spiders that come bundled with Scrapy, as well as spiders that you write yourself). It doesn't provide any special functionality. It just provides a default :meth:`start_requests` implementation which sends requests from the :attr:`start_urls` spider attribute and calls the spider's method `parse` for each of the resulting responses.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 53); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 53); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 60)

Unknown directive type "attribute".

```
.. attribute:: name

    A string which defines the name for this spider. The spider name is how
    the spider is located (and instantiated) by Scrapy, so it must be
    unique. However, nothing prevents you from instantiating more than one
    instance of the same spider. This is the most important spider attribute
    and it's required.

    If the spider scrapes a single domain, a common practice is to name the
    spider after the domain, with or without the `TLD`_. So, for example, a
    spider that crawls ``mywebsite.com`` would often be called
    ``mywebsite``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 73)

Unknown directive type "attribute".

```
.. attribute:: allowed_domains

    An optional list of strings containing domains that this spider is
    allowed to crawl. Requests for URLs not belonging to the domain names
    specified in this list (or their subdomains) won't be followed if
    :class:`~scrapy.spidermiddlewares.offsite.OffsiteMiddleware` is enabled.

    Let's say your target url is ``https://www.example.com/1.html``,
    then add ``'example.com'`` to the list.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 83)

Unknown directive type "attribute".

```
.. attribute:: start_urls

    A list of URLs where the spider will begin to crawl from, when no
    particular URLs are specified. So, the first pages downloaded will be those
    listed here. The subsequent :class:`~scrapy.Request` will be generated successively from data
    contained in the start URLs.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 90)

Unknown directive type "attribute".

```
.. attribute:: custom_settings

   A dictionary of settings that will be overridden from the project wide
   configuration when running this spider. It must be defined as a class
   attribute since the settings are updated before instantiation.

   For a list of available built-in settings see:
   :ref:`topics-settings-ref`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 99)

Unknown directive type "attribute".

```
.. attribute:: crawler

   This attribute is set by the :meth:`from_crawler` class method after
   initializating the class, and links to the
   :class:`~scrapy.crawler.Crawler` object to which this spider instance is
   bound.

   Crawlers encapsulate a lot of components in the project for their single
   entry access (such as extensions, middlewares, signals managers, etc).
   See :ref:`topics-api-crawler` to know more about them.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 110)

Unknown directive type "attribute".

```
.. attribute:: settings

   Configuration for running this spider. This is a
   :class:`~scrapy.settings.Settings` instance, see the
   :ref:`topics-settings` topic for a detailed introduction on this subject.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 116)

Unknown directive type "attribute".

```
.. attribute:: logger

   Python logger created with the Spider's :attr:`name`. You can use it to
   send log messages through it as described on
   :ref:`topics-logging-from-spiders`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 122)

Unknown directive type "attribute".

```
.. attribute:: state

   A dict you can use to persist some spider state between batches.
   See :ref:`topics-keeping-persistent-state-between-batches` to know more about it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 127)

Unknown directive type "method".

```
.. method:: from_crawler(crawler, *args, **kwargs)

    This is the class method used by Scrapy to create your spiders.

    You probably won't need to override this directly because the default
    implementation acts as a proxy to the :meth:`__init__` method, calling
    it with the given arguments ``args`` and named arguments ``kwargs``.

    Nonetheless, this method sets the :attr:`crawler` and :attr:`settings`
    attributes in the new instance so they can be accessed later inside the
    spider's code.

    :param crawler: crawler to which the spider will be bound
    :type crawler: :class:`~scrapy.crawler.Crawler` instance

    :param args: arguments passed to the :meth:`__init__` method
    :type args: list

    :param kwargs: keyword arguments passed to the :meth:`__init__` method
    :type kwargs: dict
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 148)

Unknown directive type "method".

```
.. method:: start_requests()

    This method must return an iterable with the first Requests to crawl for
    this spider. It is called by Scrapy when the spider is opened for
    scraping. Scrapy calls it only once, so it is safe to implement
    :meth:`start_requests` as a generator.

    The default implementation generates ``Request(url, dont_filter=True)``
    for each url in :attr:`start_urls`.

    If you want to change the Requests used to start scraping a domain, this is
    the method to override. For example, if you need to start by logging in using
    a POST request, you could do::

        class MySpider(scrapy.Spider):
            name = 'myspider'

            def start_requests(self):
                return [scrapy.FormRequest("http://www.example.com/login",
                                           formdata={'user': 'john', 'pass': 'secret'},
                                           callback=self.logged_in)]

            def logged_in(self, response):
                # here you would extract links to follow and return Requests for
                # each of them, with another callback
                pass
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 175)

Unknown directive type "method".

```
.. method:: parse(response)

    This is the default callback used by Scrapy to process downloaded
    responses, when their requests don't specify a callback.

    The ``parse`` method is in charge of processing the response and returning
    scraped data and/or more URLs to follow. Other Requests callbacks have
    the same requirements as the :class:`Spider` class.

    This method, as well as any other Request callback, must return an
    iterable of :class:`~scrapy.Request` and/or :ref:`item objects
    <topics-items>`.

    :param response: the response to parse
    :type response: :class:`~scrapy.http.Response`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 191)

Unknown directive type "method".

```
.. method:: log(message, [level, component])

    Wrapper that sends a log message through the Spider's :attr:`logger`,
    kept for backward compatibility. For more information see
    :ref:`topics-logging-from-spiders`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 197)

Unknown directive type "method".

```
.. method:: closed(reason)

    Called when the spider closes. This method provides a shortcut to
    signals.connect() for the :signal:`spider_closed` signal.
```

Let's see an example:

```
import scrapy


class MySpider(scrapy.Spider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = [
        'http://www.example.com/1.html',
        'http://www.example.com/2.html',
        'http://www.example.com/3.html',
    ]

    def parse(self, response):
        self.logger.info('A response from %s just arrived!', response.url)
```

Return multiple Requests and items from a single callback:

```
import scrapy

class MySpider(scrapy.Spider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = [
        'http://www.example.com/1.html',
        'http://www.example.com/2.html',
        'http://www.example.com/3.html',
    ]

    def parse(self, response):
        for h3 in response.xpath('//h3').getall():
            yield {"title": h3}

        for href in response.xpath('//a/@href').getall():
            yield scrapy.Request(response.urljoin(href), self.parse)
```

Instead of :attr:`~.start_urls` you can use :meth:`~.start_requests` directly; to give data more structure you can use :class:`~scrapy.Item` objects:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 239); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 239); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 239); *backlink*

Unknown interpreted text role "class".

```
import scrapy
from myproject.items import MyItem

class MySpider(scrapy.Spider):
    name = 'example.com'
    allowed_domains = ['example.com']

    def start_requests(self):
        yield scrapy.Request('http://www.example.com/1.html', self.parse)
        yield scrapy.Request('http://www.example.com/2.html', self.parse)
        yield scrapy.Request('http://www.example.com/3.html', self.parse)

    def parse(self, response):
        for h3 in response.xpath('//h3').getall():
            yield MyItem(title=h3)

        for href in response.xpath('//a/@href').getall():
            yield scrapy.Request(response.urljoin(href), self.parse)
```

Spider arguments

Spiders can receive arguments that modify their behaviour. Some common uses for spider arguments are to define the start URLs or to restrict the crawl to certain sections of the site, but they can be used to configure any functionality of the spider.

Spider arguments are passed through the :command:`crawl` command using the `-a` option. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 271); *backlink*

Unknown interpreted text role "command".

```
scrapy crawl myspider -a category=electronics
```

Spiders can access arguments in their *__init__* methods:

```
import scrapy

class MySpider(scrapy.Spider):
    name = 'myspider'

    def __init__(self, category=None, *args, **kwargs):
        super(MySpider, self).__init__(*args, **kwargs)
        self.start_urls = [f'http://www.example.com/categories/{category}']
        # ...
```

The default *__init__* method will take any spider arguments and copy them to the spider as attributes. The above example can also be written as follows:

```
import scrapy

class MySpider(scrapy.Spider):
    name = 'myspider'

    def start_requests(self):
        yield scrapy.Request(f'http://www.example.com/categories/{self.category}')
```

If you are :ref:`running Scrapy from a script <run-from-script>`, you can specify spider arguments when calling :class:`CrawlerProcess.crawl <scrapy.crawler.CrawlerProcess.crawl>` or :class:`CrawlerRunner.crawl <scrapy.crawler.CrawlerRunner.crawl>`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 300); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 300); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 300); *backlink*

Unknown interpreted text role "class".

```
process = CrawlerProcess()
process.crawl(MySpider, category="electronics")
```

Keep in mind that spider arguments are only strings. The spider will not do any parsing on its own. If you were to set the `start_urls` attribute from the command line, you would have to parse it on your own into a list using something like :func:`ast.literal_eval` or :func:`json.loads` and then set it as an attribute. Otherwise, you would cause iteration over a `start_urls` string (a very common python pitfall) resulting in each character being seen as a separate url.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 308); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 308); *backlink*

Unknown interpreted text role "func".

A valid use case is to set the http auth credentials used by :class:`~scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware` or the user agent used by :class:`~scrapy.downloadermiddlewares.useragent.UserAgentMiddleware`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 318); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 318); *backlink*

Unknown interpreted text role "class".

```
scrapy crawl myspider -a http_user=myuser -a http_pass=mypassword -a user_agent=mybot
```

Spider arguments can also be passed through the Scrapyd `schedule.json` API. See Scrapyd documentation.

Generic Spiders

Scrapy comes with some useful generic spiders that you can use to subclass your spiders from. Their aim is to provide convenient functionality for a few common scraping cases, like following all links on a site based on certain rules, crawling from Sitemaps, or parsing an XML/CSV feed.

For the examples used in the following spiders, we'll assume you have a project with a `TestItem` declared in a `myproject.items` module:

```
import scrapy

class TestItem(scrapy.Item):
    id = scrapy.Field()
    name = scrapy.Field()
    description = scrapy.Field()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 349)

Unknown directive type "currentmodule".

```
.. currentmodule:: scrapy.spiders
```

CrawlSpider

This is the most commonly used spider for crawling regular websites, as it provides a convenient mechanism for following links by defining a set of rules. It may not be the best suited for your particular web sites or project, but it's generic enough for several cases, so you can start from it and override it as needed for more custom functionality, or just implement your own spider.

Apart from the attributes inherited from Spider (that you must specify), this class supports a new attribute:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 365)

Unknown directive type "attribute".

```
.. attribute:: rules

    Which is a list of one (or more) :class:`Rule` objects.  Each :class:`Rule`
    defines a certain behaviour for crawling the site. Rules objects are
    described below. If multiple rules match the same link, the first one
    will be used, according to the order they're defined in this attribute.
```

This spider also exposes an overridable method:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 374)

Unknown directive type "method".

```
.. method:: parse_start_url(response, **kwargs)

   This method is called for each response produced for the URLs in
   the spider's ``start_urls`` attribute. It allows to parse
   the initial responses and must return either an
   :ref:`item object <topics-items>`, a :class:`~scrapy.Request`
   object, or an iterable containing any of them.
```

Crawling rules

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 385)

Unknown directive type "autoclass".

```
.. autoclass:: Rule

   ``link_extractor`` is a :ref:`Link Extractor <topics-link-extractors>` object which
   defines how links will be extracted from each crawled page. Each produced link will
   be used to generate a :class:`~scrapy.Request` object, which will contain the
   link's text in its ``meta`` dictionary (under the ``link_text`` key).
   If omitted, a default link extractor created with no arguments will be used,
   resulting in all links being extracted.

   ``callback`` is a callable or a string (in which case a method from the spider
   object with that name will be used) to be called for each link extracted with
   the specified link extractor. This callback receives a :class:`~scrapy.http.Response`
   as its first argument and must return either a single instance or an iterable of
   :ref:`item objects <topics-items>` and/or :class:`~scrapy.Request` objects
   (or any subclass of them). As mentioned above, the received :class:`~scrapy.http.Response`
   object will contain the text of the link that produced the :class:`~scrapy.Request`
   in its ``meta`` dictionary (under the ``link_text`` key)

   ``cb_kwargs`` is a dict containing the keyword arguments to be passed to the
   callback function.

   ``follow`` is a boolean which specifies if links should be followed from each
   response extracted with this rule. If ``callback`` is None ``follow`` defaults
   to ``True``, otherwise it defaults to ``False``.

   ``process_links`` is a callable, or a string (in which case a method from the
   spider object with that name will be used) which will be called for each list
   of links extracted from each response using the specified ``link_extractor``.
   This is mainly used for filtering purposes.

   ``process_request`` is a callable (or a string, in which case a method from
   the spider object with that name will be used) which will be called for every
   :class:`~scrapy.Request` extracted by this rule. This callable should
   take said request as first argument and the :class:`~scrapy.http.Response`
   from which the request originated as second argument. It must return a
   ``Request`` object or ``None`` (to filter out the request).

   ``errback`` is a callable or a string (in which case a method from the spider
   object with that name will be used) to be called if any exception is
   raised while processing a request generated by the rule.
   It receives a :class:`Twisted Failure <twisted.python.failure.Failure>`
   instance as first parameter.
```

```
        .. warning:: Because of its internal implementation, you must explicitly set
           callbacks for new requests when writing :class:`CrawlSpider`-based spiders;
           unexpected behaviour can occur otherwise.

        .. versionadded:: 2.0
           The *errback* parameter.
```

CrawlSpider example

Let's now take a look at an example CrawlSpider with rules:

```
import scrapy
from scrapy.spiders import CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor

class MySpider(CrawlSpider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = ['http://www.example.com']

    rules = (
        # Extract links matching 'category.php' (but not matching 'subsection.php')
        # and follow links from them (since no callback means follow=True by default).
        Rule(LinkExtractor(allow=('category\.php', ), deny=('subsection\.php', ))),

        # Extract links matching 'item.php' and parse them with the spider's method parse_item
        Rule(LinkExtractor(allow=('item\.php', )), callback='parse_item'),
    )

    def parse_item(self, response):
        self.logger.info('Hi, this is an item page! %s', response.url)
        item = scrapy.Item()
        item['id'] = response.xpath('//td[@id="item_id"]/text()').re(r'ID: (\d+)')
        item['name'] = response.xpath('//td[@id="item_name"]/text()').get()
        item['description'] = response.xpath('//td[@id="item_description"]/text()').get()
        item['link_text'] = response.meta['link_text']
        url = response.xpath('//td[@id="additional_data"]/@href').get()
        return response.follow(url, self.parse_additional_page, cb_kwargs=dict(item=item))

    def parse_additional_page(self, response, item):
        item['additional_data'] = response.xpath('//p[@id="additional_data"]/text()').get()
        return item
```

This spider would start crawling example.com's home page, collecting category links, and item links, parsing the latter with the `parse_item` method. For each item response, some data will be extracted from the HTML using XPath, and an :class:`~scrapy.Item` will be filled with it.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 473); *backlink*

Unknown interpreted text role "class".

XMLFeedSpider

XMLFeedSpider is designed for parsing XML feeds by iterating through them by a certain node name. The iterator can be chosen from: `iternodes`, `xml`, and `html`. It's recommended to use the `iternodes` iterator for performance reasons, since the `xml` and `html` iterators generate the whole DOM at once in order to parse it. However, using `html` as the iterator may be useful when parsing XML with bad markup.

To set the iterator and the tag name, you must define the following class attributes:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 493)

Unknown directive type "attribute".

```
   .. attribute:: iterator

       A string which defines the iterator to use. It can be either:

           - ``'iternodes'`` - a fast iterator based on regular expressions

           - ``'html'`` - an iterator which uses :class:`~scrapy.Selector`.
             Keep in mind this uses DOM parsing and must load all DOM in memory
             which could be a problem for big feeds

           - ``'xml'`` - an iterator which uses :class:`~scrapy.Selector`.
```

```
            Keep in mind this uses DOM parsing and must load all DOM in memory
            which could be a problem for big feeds

        It defaults to: ``'iternodes'``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 509)

Unknown directive type "attribute".

```
.. attribute:: itertag

    A string with the name of the node (or element) to iterate in. Example::

        itertag = 'product'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 515)

Unknown directive type "attribute".

```
.. attribute:: namespaces

    A list of ``(prefix, uri)`` tuples which define the namespaces
    available in that document that will be processed with this spider. The
    ``prefix`` and ``uri`` will be used to automatically register
    namespaces using the
    :meth:`~scrapy.Selector.register_namespace` method.

    You can then specify nodes with namespaces in the :attr:`itertag`
    attribute.

    Example::

        class YourSpider(XMLFeedSpider):

            namespaces = [('n', 'http://www.sitemaps.org/schemas/sitemap/0.9')]
            itertag = 'n:url'
            # ...
```

Apart from these new attributes, this spider has the following overridable methods too:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 537)

Unknown directive type "method".

```
.. method:: adapt_response(response)

    A method that receives the response as soon as it arrives from the spider
    middleware, before the spider starts parsing it. It can be used to modify
    the response body before parsing it. This method receives a response and
    also returns a response (it could be the same or another one).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 544)

Unknown directive type "method".

```
.. method:: parse_node(response, selector)

    This method is called for the nodes matching the provided tag name
    (``itertag``).  Receives the response and an
    :class:`~scrapy.Selector` for each node.  Overriding this
    method is mandatory. Otherwise, you spider won't work.  This method
    must return an :ref:`item object <topics-items>`, a
    :class:`~scrapy.Request` object, or an iterable containing any of
    them.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 554)

Unknown directive type "method".

```
.. method:: process_results(response, results)

    This method is called for each result (item or request) returned by the
    spider, and it's intended to perform any last time processing required
    before returning the results to the framework core, for example setting the
    item IDs. It receives a list of results and the response which originated
    those results. It must return a list of results (items or requests).
```

Warning

Because of its internal implementation, you must explicitly set callbacks for new requests when writing :class:`XMLFeedSpider`-based spiders; unexpected behaviour can occur otherwise.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 562); *backlink*

Unknown interpreted text role "class".

XMLFeedSpider example

These spiders are pretty easy to use, let's have a look at one example:

```
from scrapy.spiders import XMLFeedSpider
from myproject.items import TestItem

class MySpider(XMLFeedSpider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = ['http://www.example.com/feed.xml']
    iterator = 'iternodes'  # This is actually unnecessary, since it's the default value
    itertag = 'item'

    def parse_node(self, response, node):
        self.logger.info('Hi, this is a <%s> node!: %s', self.itertag, ''.join(node.getall()))

        item = TestItem()
        item['id'] = node.xpath('@id').get()
        item['name'] = node.xpath('name').get()
        item['description'] = node.xpath('description').get()
        return item
```

Basically what we did up there was to create a spider that downloads a feed from the given `start_urls`, and then iterates through each of its `item` tags, prints them out, and stores some random data in an :class:`~scrapy.Item`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 591); *backlink*

Unknown interpreted text role "class".

CSVFeedSpider

This spider is very similar to the XMLFeedSpider, except that it iterates over rows, instead of nodes. The method that gets called in each iteration is :meth:`parse_row`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 600); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 604)

Unknown directive type "attribute".

```
.. attribute:: delimiter

    A string with the separator character for each field in the CSV file
    Defaults to ``','`` (comma).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 609)

Unknown directive type "attribute".

```
.. attribute:: quotechar

    A string with the enclosure character for each field in the CSV file
    Defaults to ``'"'`` (quotation mark).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 614)

Unknown directive type "attribute".

```
.. attribute:: headers

    A list of the column names in the CSV file.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 618)

Unknown directive type "method".

```
.. method:: parse_row(response, row)

    Receives a response and a dict (representing each row) with a key for each
    provided (or detected) header of the CSV file.  This spider also gives the
    opportunity to override ``adapt_response`` and ``process_results`` methods
    for pre- and post-processing purposes.
```

CSVFeedSpider example

Let's see an example similar to the previous one, but using a :class:`CSVFeedSpider`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 628); *backlink*

Unknown interpreted text role "class".

```
from scrapy.spiders import CSVFeedSpider
from myproject.items import TestItem

class MySpider(CSVFeedSpider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = ['http://www.example.com/feed.csv']
    delimiter = ';'
    quotechar = "'"
    headers = ['id', 'name', 'description']

    def parse_row(self, response, row):
        self.logger.info('Hi, this is a row!: %r', row)

        item = TestItem()
        item['id'] = row['id']
        item['name'] = row['name']
        item['description'] = row['description']
        return item
```

SitemapSpider

SitemapSpider allows you to crawl a site by discovering the URLs using Sitemaps.

It supports nested sitemaps and discovering sitemap urls from robots.txt.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 663)

Unknown directive type "attribute".

```
.. attribute:: sitemap_urls

    A list of urls pointing to the sitemaps whose urls you want to crawl.
```

```
    You can also point to a `robots.txt`_ and it will be parsed to extract
    sitemap urls from it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 670)

Unknown directive type "attribute".

```
.. attribute:: sitemap_rules

    A list of tuples ``(regex, callback)`` where:

    * ``regex`` is a regular expression to match urls extracted from sitemaps.
      ``regex`` can be either a str or a compiled regex object.

    * callback is the callback to use for processing the urls that match
      the regular expression. ``callback`` can be a string (indicating the
      name of a spider method) or a callable.

    For example::

        sitemap_rules = [('/product/', 'parse_product')]

    Rules are applied in order, and only the first one that matches will be
    used.

    If you omit this attribute, all urls found in sitemaps will be
    processed with the ``parse`` callback.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 691)

Unknown directive type "attribute".

```
.. attribute:: sitemap_follow

    A list of regexes of sitemap that should be followed. This is only
    for sites that use `Sitemap index files`_ that point to other sitemap
    files.

    By default, all sitemaps are followed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 699)

Unknown directive type "attribute".

```
.. attribute:: sitemap_alternate_links

    Specifies if alternate links for one ``url`` should be followed. These
    are links for the same website in another language passed within
    the same ``url`` block.

    For example::

        <url>
            <loc>http://example.com/</loc>
            <xhtml:link rel="alternate" hreflang="de" href="http://example.com/de"/>
        </url>

    With ``sitemap_alternate_links`` set, this would retrieve both URLs. With
    ``sitemap_alternate_links`` disabled, only ``http://example.com/`` would be
    retrieved.

    Default is ``sitemap_alternate_links`` disabled.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)spiders.rst, line 718)

Unknown directive type "method".

```
.. method:: sitemap_filter(entries)
```

```
        This is a filter function that could be overridden to select sitemap entries
        based on their attributes.

        For example::

            <url>
                <loc>http://example.com/</loc>
                <lastmod>2005-01-01</lastmod>
            </url>

        We can define a ``sitemap_filter`` function to filter ``entries`` by date::

            from datetime import datetime
            from scrapy.spiders import SitemapSpider

            class FilteredSitemapSpider(SitemapSpider):
                name = 'filtered_sitemap_spider'
                allowed_domains = ['example.com']
                sitemap_urls = ['http://example.com/sitemap.xml']

                def sitemap_filter(self, entries):
                    for entry in entries:
                        date_time = datetime.strptime(entry['lastmod'], '%Y-%m-%d')
                        if date_time.year >= 2005:
                            yield entry

        This would retrieve only ``entries`` modified on 2005 and the following
        years.

        Entries are dict objects extracted from the sitemap document.
        Usually, the key is the tag name and the value is the text inside it.

        It's important to notice that:

        - as the loc attribute is required, entries without this tag are discarded
        - alternate links are stored in a list with the key ``alternate``
          (see ``sitemap_alternate_links``)
        - namespaces are removed, so lxml tags named as ``{namespace}tagname`` become only ``tagname``

        If you omit this method, all entries found in sitemaps will be
        processed, observing other attributes and their settings.
```

SitemapSpider examples

Simplest example: process all urls discovered through sitemaps using the `parse` callback:

```
from scrapy.spiders import SitemapSpider

class MySpider(SitemapSpider):
    sitemap_urls = ['http://www.example.com/sitemap.xml']

    def parse(self, response):
        pass # ... scrape item here ...
```

Process some urls with certain callback and other urls with a different callback:

```
from scrapy.spiders import SitemapSpider

class MySpider(SitemapSpider):
    sitemap_urls = ['http://www.example.com/sitemap.xml']
    sitemap_rules = [
        ('/product/', 'parse_product'),
        ('/category/', 'parse_category'),
    ]

    def parse_product(self, response):
        pass # ... scrape product ...

    def parse_category(self, response):
        pass # ... scrape category ...
```

Follow sitemaps defined in the robots.txt file and only follow sitemaps whose url contains `/sitemap_shop`:

```
from scrapy.spiders import SitemapSpider

class MySpider(SitemapSpider):
    sitemap_urls = ['http://www.example.com/robots.txt']
    sitemap_rules = [
        ('/shop/', 'parse_shop'),
    ]
    sitemap_follow = ['/sitemap_shops']
```

```
        def parse_shop(self, response):
            pass # ... scrape shop here ...
```

Combine SitemapSpider with other sources of urls:

```
    from scrapy.spiders import SitemapSpider

    class MySpider(SitemapSpider):
        sitemap_urls = ['http://www.example.com/robots.txt']
        sitemap_rules = [
            ('/shop/', 'parse_shop'),
        ]

        other_urls = ['http://www.example.com/about']

        def start_requests(self):
            requests = list(super(MySpider, self).start_requests())
            requests += [scrapy.Request(x, self.parse_other) for x in self.other_urls]
            return requests

        def parse_shop(self, response):
            pass # ... scrape shop here ...

        def parse_other(self, response):
            pass # ... scrape other here ...
```