Jobs: pausing and resuming crawls

Sometimes, for big sites, it's desirable to pause crawls and be able to resume them later.

Scrapy supports this functionality out of the box by providing the following facilities:

- a scheduler that persists scheduled requests on disk
- a duplicates filter that persists visited requests on disk
- an extension that keeps some spider state (key/value pairs) persistent between batches

Job directory

To enable persistence support you just need to define a *job directory* through the `JOBDIR` setting. This directory will be for storing all required data to keep the state of a single job (i.e. a spider run). It's important to note that this directory must not be shared by different spiders, or even different jobs/runs of the same spider, as it's meant to be used for storing the state of a *single* job.

How to use it

To start a spider with persistence support enabled, run it like this:

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

Then, you can stop the spider safely at any time (by pressing Ctrl-C or sending a signal), and resume it later by issuing the same command:

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

Keeping persistent state between batches

Sometimes you'll want to keep some persistent spider state between pause/resume batches. You can use the `spider.state` attribute for that, which should be a dict. There's a built-in extension that takes care of serializing, storing and loading that attribute from the job directory, when the spider starts and stops.

Here's an example of a callback that uses the spider state (other spider code is omitted for brevity):

```
def parse_item(self, response):
    # parse item here
    self.state['items_count'] = self.state.get('items_count', 0) + 1
```

Persistence gotchas

There are a few things to keep in mind if you want to be able to use the Scrapy persistence support:

Cookies expiration

Cookies may expire. So, if you don't resume your spider quickly the requests scheduled may no longer work. This won't be an issue if your spider doesn't rely on cookies.

Request serialization

For persistence to work, :class:`~scrapy.Request` objects must be serializable with :mod:`pickle`, except for the `callback` and `errback` values passed to their `__init__` method, which must be methods of the running :class:`~scrapy.Spider` class.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)jobs.rst, line 79); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)jobs.rst, line 79); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)jobs.rst, line 79); *backlink*

Unknown interpreted text role "class".

If you wish to log the requests that couldn't be serialized, you can set the :setting:`SCHEDULER_DEBUG` setting to `True` in the project's settings page. It is `False` by default.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\(scrapy-master)(docs)(topics)jobs.rst`, **line 84**); ***backlink***
>
> Unknown interpreted text role "setting".