Scrapy Python web crawling

CONCEPTS

Develop selectors in scrapy shell first, then paste them into the spider's parse().

Spider start requests
↓ Request
Downloader fetches page
↓ Response
parse() callback
↓ yield item
Item pipeline / feed clean, store

Commands marked project-only (crawl, list, check, parse, edit) must run inside a folder with scrapy.cfg.

Quote URLs in the shell: an unquoted & ends the command.

PROJECT SETUP

`scrapy startproject <name> [dir]`
Create a project skeleton

`scrapy genspider <name> <domain>`
Create a spider from a template

`scrapy genspider -l`
List spider templates

`scrapy genspider -t crawl <name> <domain>`
Spider with link-following rules

`scrapy list`
List spiders in this project

RUNNING SPIDERS

`scrapy crawl <spider>`
Run a project spider

`scrapy runspider <file.py>`
Run a standalone spider file

`scrapy crawl <spider> -a <name>=<value>`
Pass a spider argument

`scrapy crawl <spider> -s <NAME>=<value>`
Override a setting once

`scrapy crawl <spider> -s DOWNLOAD_DELAY=1`
Slow down requests

`scrapy crawl <spider> -L INFO`
Quieter log level

RUNNING SPIDERS (cont.)

`scrapy crawl <spider> --nolog`
Disable logging

`scrapy crawl <spider> -s JOBDIR=<dir>`
Pause and resume a crawl

OUTPUT (FEEDS)

`scrapy crawl <spider> -o <file>`
Append items to a file

`scrapy crawl <spider> -O <file>`
Overwrite the file with items

`scrapy crawl <spider> -o <file>:<format>`
Force an output format

`scrapy crawl <spider> -O out.jsonl`
One JSON object per line

`scrapy crawl <spider> -O out.csv`
Write CSV

SHELL AND INSPECTION

`scrapy shell "<url>"`
Open the console on a page

`scrapy shell -c "response.css('title::text').get()" "<url>"`
One-shot expression

`scrapy fetch "<url>"`
Print the body as Scrapy sees it

`scrapy fetch --headers "<url>"`
Print response headers instead

`scrapy view "<url>"`
Open the page as Scrapy got it

`scrapy parse --spider=<name> "<url>"`
Run the spider's callback on one URL

`scrapy parse -c <callback> "<url>"`
Choose the callback

`scrapy settings --get <SETTING>`
Print one setting's value

SHELL OBJECTS AND SELECTORS

`response.css("<sel>")`
CSS selector on the page

`response.css("<sel>::text").getall()`
All matching text nodes

`response.css("<sel>::attr(href)").get()`
First attribute value or None

`response.xpath("//<tag>")`
XPath selector on the page

`view(response)`
Open the response in a browser

`fetch("<url>")`
Load another page into the shell

`inspect_response(response, self)`
Open a shell inside a running spider

SPIDER CODE

`yield {"<key>": <value>}`
Emit an item from parse()

`yield response.follow(<href>, callback=self.<fn>)`
Follow a link to another callback

`response.follow_all(css="<sel>", callback=self.<fn>)`
Follow every matching link

`yield scrapy.Request(<url>, callback=self.<fn>)`
Queue an explicit request

`start_urls = ["<url>"]`
Pages the spider begins on

RECIPES

Minimal standalone spider

```
# quotes.py
import scrapy
class Quotes(scrapy.Spider):
    name = "quotes"
    start_urls = ["https://
quotes.toscrape.com"]
    def parse(self, response):
        for q in
response.css("div.quote"):
            yield {"text":
q.css("span.text::text").get()}
        yield from
response.follow_all(css="li.next
a", callback=self.parse)

scrapy runspider quotes.py -O
quotes.jsonl
```