Scrapy 基本概念

前文介绍了Scrapy的基本使用，为了深入掌握Scrapy这一强大爬虫工具，需要学习它的一些概念。

Command line tool

我们已经知道Scrapy是由scrapy命令行控制的，在命令行中运行scrapy后，会看到许多子命令，我们把他们统称为scrapy命令。

配置文件

在执行scrapy命令时，Scrapy会参考配置文件scrapy.cfg中的内容进行设置，这个文件位于每个Scrapy项目的顶级目录内。事实上，Scrapy的全局配置和用户配置文件是每个项目中的配置文件的基础文件，他们位于：

```
/etc/scrapy.cfg or c:\scrapy\scrapy.cfg (system-wide)
~/.config/scrapy.cfg ($XDG_CONFIG_HOME) and ~/.scrapy.cfg ($HOME)
for global (user-wide) settings, and
scrapy.cfg inside a scrapy project’s root (see next section).
```

Scrapy 项目

Scrapy项目的默认结构如下：

```
scrapy.cfg
myproject/
    __init__.py
    items.py
    middlewares.py
    pipelines.py
    settings.py
    spiders/
        __init__.py
        spider1.py
        spider2.py
        ...
```

新建项目的基本流程是：

1. 如果要新建自定义的Scrapy项目，首先要在命令行运行命令：

    > scrapy startproject myproject [project_dir]

1. 之后进入项目目录project_dir

    > cd project_dir

1. 生成新的自定义爬虫文件

> scrapy genspider mydomain mydomain.com

1. 初步运行爬虫

首先要进入项目所在目录，之后运行：

> scrapy crawl mydomain

全局**Scrapy**命令

Scrapy提供的全局命令有：

- startproject
  - 语法：crapy startproject [project_dir]
  - 功能：用于生成新的自定义Scrapy项目
- genspider
  - 语法：scrapy genspider [-t template]
  - 功能：在当前目录或当前项目的spiders目录中生成新的爬虫文件；
  - 说明：可以使用scrapy genspider -l查看爬虫模板，目前模板类型有basic、crawl、csvfeed、xmlfeed。
  - 举例：scrapy genspider example example.com 或 scrapy genspider -t crawl scrapyorg scrapy.org
- settings
  - 语法：scrapy shell
  - 功能：设置Scrapy配置项
- runspider
  - 语法：scrapy runspider <spider_file.py>
  - 功能：单独运行一个python文件中包含的爬虫，而不生成一个Scrapy项目。
- shell
  - 语法：scrapy shell
  - 功能：爬取url指定的网页，并启动Scrapy shell。
  - 支持选项：
    - --spider=SPIDER: bypass spider autodetection and force use of specific spider
    - -c code: evaluate the code in the shell, print the result and exit
    - --no-redirect: do not follow HTTP 3xx redirects (default is to follow them); this only affects the URL you may pass as argument on the command line; once you are inside the shell, fetch(url) will still follow HTTP redirects by default.
- fetch
  - 语法：scrapy fetch
  - 功能：使用Scrapy downloader下载给定的URL，并将文件写到标准输出中。常用于编写爬虫文件前的试探工作。
  - 支持的选项：
    - --spider=SPIDER: bypass spider autodetection and force use of specific spider
    - --headers: print the response’s HTTP headers instead of the response’s body
    - --no-redirect: do not follow HTTP 3xx redirects (default is to follow

them)

- view
  - 语法：scrapy view
  - 功能：在浏览器中打开给定URL指定的网页。
  - 可选参数：
    - --spider=SPIDER: bypass spider autodetection and force use of specific spider
    - --no-redirect: do not follow HTTP 3xx redirects (default is to follow them)
- version
  - 语法：scrapy version [-v]
  - 功能：查看scrapy版本

项目级命令

- crawl
  - 语法：scrapy crawl
  - 功能：运行爬虫spider
- check
  - 语法：scrapy check [-l]
  - 功能：运行Scrapy合规检查
- list
  - 语法：scrapy list
  - 功能：列出当前项目中所有可用的爬虫文件
- edit
  - 语法：scrapy edit
  - 功能：使用配置文件中指定的编辑器去编辑爬虫文件spider
- parse
  - 语法：scrapy parse [options]
  - 功能：获取url指定页面并使用指定的爬虫来解析，默认的解析函数名为parse。
  - 可选的参数：
    - --spider=SPIDER: bypass spider autodetection and force use of specific spider
    - --a NAME=VALUE: set spider argument (may be repeated)
    - --callback or -c: spider method to use as callback for parsing the response
    - --meta or -m: additional request meta that will be passed to the callback request. This must be a valid json string. Example: –meta=’{“foo” : “bar”}’
    - --pipelines: process items through pipelines
    - --rules or -r: use CrawlSpider rules to discover the callback (i.e. spider method) to use for parsing the response
    - --noitems: don’t show scraped items
    - --nolinks: don’t show extracted links
    - --nocolour: avoid using pygments to colorize the output
    - --depth or -d: depth level for which the requests should be followed recursively (default: 1)
    - --verbose or -v: display information for each depth level
- bench
  - 语法：scrapy bench
  - 功能：运行scrapy快速性能测试。

自定义项目命令

用户可以自定义项目级Scrapy命令，只需要在COMMANDS_MODULE设置项中进行设置。具体情况请参考：https://docs.scrapy.org/en/latest/topics/commands.html

Spiders

Spiders是定义如何爬取某些网站的python类。Spiders包含内容：

- 如何执行页面爬取？
- 如何提取结构化的数据？

Spider类的基本处理流程：

1. 使用start_urls列表设置初始URLs，生成初始化爬取请求；
    - 初始URLs的提供可以使用start_urls列表，也可以使用start_requests()方法。
2. 在回调函数中，使用parse方法来解析响应，并返回提取数据的结构：字典对象、Item对象、Request对象或其他类型的可迭代对象；
3. 在回调函数中，通常使用Selectors对象定位元素和提取数据，也可以使用BeautifulSoup、lxml或其他机制；
4. 最后，由spider返回的items将被持久化到数据库或文件中。

scrapy.spiders.Spider的属性和方法

属性

- name：必须项，一个spider类/对象的名字，是spider类在scrapy项目中的唯一标识。
- allowed_domains：可选项，用于设置spider爬取网站时的域名约束，不属于该域的URLs将不被爬取，当然OffsiteMiddleware应设为enabled。
- start_urls：必须项，spider对象的初始爬取链接。
- custom_settings：设置字典
- crawler
- settings
- logger

方法

- from_crawler(crawler, *args, **kwargs）
- start_requests()
- parse(response)
- log(message[, level, component])
- closed(reason)

In [ ]:
```
"""示例1: Spider的简单例子"""
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.ur
l)
```

In [ ]:
```
"""示例2: 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').extract():
            yield {"title": h3}

        for url in response.xpath('//a/@href').extract():
            yield scrapy.Request(url, callback=self.parse)
```

In [ ]:
```
"""示例3: 可以使用start_requests()代替start_urls"""
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').extract():
            yield MyItem(title=h3)

        for url in response.xpath('//a/@href').extract():
            yield scrapy.Request(url, callback=self.parse)
```

Spider参数

Spiders可以接收命令行参数，由此改变自身行为。一些常见的Spider参数用于定义起始URLs，或限制爬取某个特定域名。

运行参数设置需要使用"-a 参数=值"的命令格式，例如：

> scrapy crawl myspider -a category=electronics

相应地，在Spider类中应有如下例的定义：

```
import scrapy

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

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

初始化方法将接收命令行传入的参数，将其拷贝为spider对象的属性。

注意：所有的参数都是字符串格式的。Spider并不会自动将其转换为其他类型对象。

通用爬虫

Scrapy提供了一些有用的通用爬虫，可以根据这些通用爬虫生成自己的子类。这些通用爬虫可以为通用爬取任务提供一些便利的方法，例如在一个网站内遍历链接，从Sitemaps爬取所有链接，或解析XML/CSV数据。

常用的通用爬虫有：

CrawlSpider

> class scrapy.spiders.CrawlSpider

这是一个最为常用的Spider，常用于爬取规范网站，根据规则遍历链接。CrawlSpider有两个新的属性/方法：

- rules
  - 一个Rule对象的列表或一个Rule对象
  - 每个Rule对象定义了一定的爬取行为，如果多个rules都匹配了同一个url，那么第一个定义的将被使用。
- parse_start_url(response)
  - 一个可重载的方法
  - 该方法将被start_urls响应调用，它允许解析初始响应对象，并且必须返回一个Item对象或一个Request队形，或包含他们的一个可迭代对象。

rules

在rules中包含一个或多个Rule对象，每个Rule对爬取网站的动作定义了特定操作。如果多个rule匹配了相同的链接，则根据规则在本集合中被定义的顺序，第一个会被使用。

```
class scrapy.spiders.Rule(
        link_extractor,
        callback = None,
        cb_kwargs = None,
        follow = None,
        process_links = None,
        process_request = None
```

```
)
```

主要参数:

- link_extractor：是一个Link Extractor对象，用于定义需要提取的链接。
- callback： 从link_extractor中每获取到链接时，参数所指定的值作为回调函数，该回调函数接受一个response作为其第一个参数。
  - 注意：当编写爬虫规则时，避免使用parse作为回调函数。由于CrawlSpider使用parse方法来实现其逻辑，如果覆盖了 parse方法，crawl spider将会运行失败。
- follow：是一个布尔(boolean)值，指定了根据该规则从response提取的链接是否需要跟进。 如果callback为None，follow 默认设置为True ，否则默认为False。
- process_links：指定该spider中哪个的函数将会被调用，从link_extractor中获取到链接列表时将会调用该函数。该方法主要用来过滤。
- process_request：指定该spider中哪个的函数将会被调用， 该规则提取到每个request时都会调用该函数。 (用来过滤request)

link_extractor

class scrapy.linkextractors.LinkExtractor

Link Extractors 的目的很简单: 提取链接

每个LinkExtractor有唯一的公共方法是 extract_links()，它接收一个 Response 对象，并返回一个 scrapy.link.Link 对象。

Link Extractors要实例化一次，并且 extract_links 方法会根据不同的 response 调用多次提取链接

```
class scrapy.linkextractors.LinkExtractor(
    allow = (),
    deny = (),
    allow_domains = (),
    deny_domains = (),
    deny_extensions = None,
    restrict_xpaths = (),
    tags = ('a','area'),
    attrs = ('href'),
    canonicalize = True,
    unique = True,
    process_value = None
)
```

主要参数:

- allow：满足括号中“正则表达式”的值会被提取，如果为空，则全部匹配。
- deny：与这个正则表达式(或正则表达式列表)不匹配的URL一定不提取。
- allow_domains：会被提取的链接的domains。
- deny_domains：一定不会被提取链接的domains。
- restrict_xpaths：使用xpath表达式，和allow共同作用过滤链接。

使用命令行新建**CrawlSpider**

> scrapy genspider -t crawl quotescrawlspider quotes.toscrape.com

In [ ]:
```
"""CrawlSpider 示例"""

# -*- coding: utf-8 -*-
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule


class QuotescrawlspiderSpider(CrawlSpider):
    name = 'quotescrawlspider'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['http://quotes.toscrape.com/']

    rules = (
        # Extract links matching 'item.php' and parse them with the spid
er's method parse_item
        Rule(LinkExtractor(allow=r'Items/'), callback='parse_item', foll
ow=True),
         # Extract links matching 'category.php' (but not matching 'subs
ection.php')
        # and follow links from them (since no callback means follow=Tru
e by default).
        Rule(LinkExtractor(allow=('category\.php', ), deny=('subsection\
.php', ))),
    )

    def parse_item(self, response):
        i = {}
        #i['domain_id'] = response.xpath('//input[@id="sid"]/@value').ex
tract()
        #i['name'] = response.xpath('//div[@id="name"]').extract()
        #i['description'] = response.xpath('//div[@id="description"]').e
xtract()
        return i
```

爬取规则实例

1. 运行

> scrapy shell "http://hr.tencent.com/position.php?&start=0#a"

1. 在spider文件中导入LinkExtractor，创建LinkExtractor实例对象;

> from scrapy.linkextractors import LinkExtractor
>
> page_lx = LinkExtractor(allow=('position.php?&start=\d+'))

在这里:

- allow : LinkExtractor对象最重要的参数之一，这是一个正则表达式，必须要匹配这个正则表达式(或正则表达式列表)的URL才会被提取，如果没有给出(或为空), 它会匹配所有的链接
- deny : 用法同allow，只不过与这个正则表达式匹配的URL不会被提取)　它的优先级高于 allow 的参数，如果没有给出(或None), 将不排除任何链接

1. 调用LinkExtractor实例的extract_links()方法查询匹配结果:

> page_lx.extract_links(response) 结果为: [] 修改(注意转义字符)上面语句： page_lx = LinkExtractor(allow=('position.php\?&start=\d+'))
>
> # page_lx = LinkExtractor(allow = ('start=\d+'))
>
> page_lx.extract_links(response)

1. 测试成功后，修改spdier代码

代码示例：

```
#提取匹配 'http://hr.tencent.com/position.php?&start=\d+'的链接
page_lx = LinkExtractor(allow = ('start=\d+'))

rules = [
    #提取匹配,并使用spider的parse方法进行分析;并跟进链接(没有callback意味着follow默认为True)
    Rule(page_lx, callback = 'parse', follow = True)
]
```

上面的代码事实上是错误的，callback处不能设置为parse，因为parse是CrawlSpider实现自身逻辑的，所以用户借用模板生成的spider就不能重载这函数了，否则执行会失败。

1. 再次修改spider代码

In [ ]:
```
import scrapy
from scrapy.spiders import CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor
from mySpider.items import TencentItem

class TencentSpider(CrawlSpider):
    name = "tencent"
    allowed_domains = ["hr.tencent.com"]
    start_urls = [
        "http://hr.tencent.com/position.php?&start=0#a"
    ]

    page_lx = LinkExtractor(allow=("start=\d+"))

    rules = [
        Rule(page_lx, callback = "parseContent", follow = True)
    ]

    def parseContent(self, response):
        for each in response.xpath('//table[@class="tablelist"]/tr'):
            name = each.xpath('./td[1]/a/text()').extract()[0]
            detailLink = each.xpath('./td[1]/a/@href').extract()[0]
            positionInfo = each.xpath('./td[2]/text()').extract()[0]

            peopleNumber = each.xpath('./td[3]/text()').extract()[0]
            workLocation = each.xpath('./td[4]/text()').extract()[0]
            publishTime = each.xpath('./td[5]/text()').extract()[0]
            #print name, detailLink, catalog,recruitNumber,workLocation,publishTime
```

```
            item = TencentItem()
            item['name']=name
            item['detailLink']=detailLink
            item['positionInfo']=positionInfo
            item['peopleNumber']=peopleNumber
            item['workLocation']=workLocation
            item['publishTime']=publishTime

            yield item

    # parse() 方法不需要重写
    # def parse(self, response):

    #     pass
```

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 Item will be filled with it.

XMLFeedSpider

> class scrapy.spiders.XMLFeedSpider

XMLFeedSpider类是为解析XML数据而专门设计的Spider类。这个类处理XML的方法是将XML节点名迭代地进行处理，迭代器可以是iternodes、ml和html。推荐使用iternodes作为迭代器，这样效率会更高。而XML和HTML将解析全文档为DOM对象，这对于较大的XML或HTML文档的处理将会没有效率。

为设置迭代器（iterator ）和标签名（tag），我们需要定义下列类属性:

- iterator
  - 定义迭代器时使用的字符串。可以是:
  - 'iternodes'
  - 'html'
  - 'xml'
- itertag
  - 迭代节点的名字，例如：itertag = "product"
- namespace
  - 一个（prefix，uri）元组组成的列表。例如:

```
class YourSpider(XMLFeedSpider):

    namespaces = [('n', 'http://www.sitemaps.org/schemas/sitemap/0
.9')]
    itertag = 'n:url'
    # ...
```

- 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).
- parse_node(response, selector)

    - This method is called for the nodes matching the provided tag name (itertag). Receives the response and an Selector for each node. Overriding this method is mandatory. Otherwise, you spider won't work. This method must return either a Item object, a Request object, or an iterable containing any of them.
- 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).

In [ ]:
```
"""XMLFeedSpider 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 t
he default value
    itertag = 'item'

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

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

CSVFeedSpider

> class scrapy.spiders.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 parse_row().

- delimiter
    - A string with the separator character for each field in the CSV file Defaults to ',' (comma).
- quotechar
    - A string with the enclosure character for each field in the CSV file Defaults to "" (quotation mark).
- headers
    - A list of the column names in the CSV file.
- 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.

In [ ]:
```
"""CSVFeedSpider example"""
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

> class scrapy.spiders.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.

- 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.
- 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.

- sitemap_follow
  - A list of regexes of sitemap that should be followed. This is is only for sites that use Sitemap index files that point to other sitemap files.
  - By default, all sitemaps are followed.
- 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://exampl
e.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.

In [ ]:
```
"""SitemapSpider examples"""
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 ...
```

Item

爬虫的主要目标是从无结构的数据源中爬取结构化数据。 Scrapy通常以字典形式返回爬取得数据。然而python字典的结构性并不好。

为了定义更为通用的输出数据格式，Scrapy提供了Item类。Item对象是用于收集爬取得数据的简单容器。它提供类似字典的语法来访问内部fields。

Scrapy还通过Items提供许多其他的功能：

- exporter，通过查看已定义的Item属性，给出输出的列；
- seralization可根据fields元数据进行自定义；
- trackref可以跟踪tracks Item实例，帮助发现内存泄露。

声明Items

定义Items需要使用一个简单类定义语法和Field对象，例如：

In [ ]:
```
import scrapy

class Product(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field()
    stock = scrapy.Field()
    last_updated = scrapy.Field(serializer=str)
```

Item Fields

Field对象被用于定义每个item field的元数据。

下面给出一些常见的item定义和用法：

In [11]:
```
import scrapy

class Product(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field()
    stock = scrapy.Field()
    last_updated = scrapy.Field(serializer=str)

# 生成Product item对象
product = Product(name = 'Desktop PC', price = 1000)
print(product)
# 使用get方法获取指定field的值
print(product.get('name'))
# 对于没有定义的field，使用get方法较为安全
print(product.get('last_updated'))
# 判断某个field name是否在对象中
if 'name' in product:
    print(True)
# 判断某个字符串是否为item对象的field
'lala' in product.fields

# 设置field 值,field必须已经定义，未定义的field必须先定义。
product['last_updated'] = 'today'
print(product.get('last_updated'))

# 访问所有赋值
print(product.keys())

print(product.items())
```

```
{'name': 'Desktop PC', 'price': 1000}
Desktop PC
None
True
today
dict_keys(['name', 'price', 'last_updated'])
ItemsView({'last_updated': 'today', 'name': 'Desktop PC', 'price': 1000}
)
```

Item Pipeline

当Item在Spider中被收集之后，它将会被传递到Item Pipeline，这些Item Pipeline组件按定义的顺序处理Item。

每个Item Pipeline都是实现了简单方法的Python类，比如决定此Item是丢弃而存储。以下是item pipeline的一些典型应用：

- cleansing HTML data
- validating scraped data (checking that the items contain certain fields)
- checking for duplicates (and dropping them)
- storing the scraped item in a database

Item pipeline实现

编写Item pipeline较为简单，Item pipeline组件是一个独立的Python类，其中process_item()方法必须实现：

In [ ]:
```
"""Item pipeline程序模板示例"""
class myfirstPipeline(object):
    def __init__(self):
        """TODO 参数初始化"""
        pass

    def process_item(self,item,spider):
        """
        功能：item处理函数，必须
        参数：
            item：待处理的已爬取item
            spider：item的制造者spider
        返回：
            一个Item对象（对原item进行增删改等处理后的包装对象）
        """
        return item
    def open_spider(self, spider):
        """
        功能：当参数spider指定的爬虫运行时，执行本函数。
        参数：
            spider：被开启的spider

        """
    def close_spider(self, spider):
        """
        功能：当参数spider指定的爬虫关闭时，执行本函数。
        参数：
            spider：被关闭的spider

        """
    @classmethod
    def from_crawler(cls,crawler):
        """
        功能：如果重载该函数，这个类方法调用时，会根据一个Crawler生成一个pipeline实例。
        参数：
            Crawler：使用这个pipeline的爬虫。Crawler对象提供了对所有Scrapy核心组件的访问，例如settings和signals。这是一种pipeline访问其他scrapy项目资源的方式。

        返回：必须返回一个pipeline实例。
        """
```

ItemLoader

ItemLoader是Scrapy中设计的一套用于处理已爬取Items的简便模式。虽然Items可以使用其他类似的字典API函数，但ItemLoaders提供了更为方便的处理方法。我们可以使用它完成一定的数据的清洗工作。

可以这样理解：

- Items类是爬得数据的容器；

ItemLoaders类是处理容器的机制/方法。

为了使用 ItemLoaders 处理items，需要以下步骤：

1. 初始化ItemLoaders，你可以使用一个字典类对象初始化；
2. 收集有价值的数据，将其追加放入item loader,典型的方法是使用Selector；
3. 进一步的数据清洗

下面是一个典型的使用ItemLoader的例子：

In [ ]:
```
"""使用Itemloader处理items的示例"""
from scrapy.loader import ItemLoader
from myproject.items import Product

def parse(self, response):
    l = ItemLoader(item=Product(), response=response)
    l.add_xpath('name', '//div[@class="product_name"]')
    l.add_xpath('name', '//div[@class="product_title"]')
    l.add_xpath('price', '//p[@id="price"]')
    l.add_css('stock', 'p#stock]')
    l.add_value('last_updated', 'today') # you can also use literal valu
es
    return l.load_item()
```

可以看出，上面例子中的name字段使用了两种XPATH进行提取，这两种方法会获取两个不同的name值，这两个值都将被保留。

ItemLoader.load_item()方法用于返回一个item对象，而parse函数最后将返回这个对象。

输入输出处理器

每个ItemLoader对象都包含一个处理每个item field的输入处理器（processor）和一个输出processor。

- 输入processor，将在itemloader对象一接收到item时就调用，例如上例中，将在add_xxx方法执行后就自动调用；处理结果被收集和保存在ItemLoader对象中。
- 输出processor，将在调用load_item()后被执行，操作可能会涉及到之前所有或部分已收集到的item，处理结果将作为item对象最后的结果。

数据清洗工作可以由processor完成，常用的processor有：

- Join()
  - 用于将多个结果合并为一个
- MapCompose(unicode.strip)
  - 用于一处前置或后置的空白字符
- Mapcompose(unicode.strip, unicode.title)
  - 类似MapCompose(unicode.strip)，同时提供类标题结果
- MapCompose(float)
  - 将字符串转换为数字
- MapCompose(lambda i:i.replace(',', ''), float)
  - 将字符串转换为数字，同时忽略可能的‘，’字符
- MapCompose(lambda i:urlparse.urljoin(reponse.url,i))
  - 将相关的URLs转化为绝对URLs，使用response.url为基本URL。

实现JSON文件输出

In [ ]:
```
"""完善前文tutorial示例中的item处理

之前的scrapy tutorial项目中，对爬取得到的数据仅仅是输出为json文件，
"""
"""
以下pipeline将所有(从所有'spider'中)爬取到的item，
存储到一个独立地items.json 文件，每行包含一个序列化为'JSON'格式的'item'
"""
import json

class Tutorial01Pipeline(object):
    def __init__(self):
        self.file = open('quotes1.json','wb')

    def process_item(self, item, spider):
        content = json.dumps(dict(item),ensure_ascii = False) + '\n'
        self.file.write(content)

        return item

    def close_spider(self,spider):
        self.file.close()
```

启用一个Item pipeline组件

为了启用一个item pipeline组件，必须将它的类名添加导settings.py文件ITEM_PIPELINES中。

例如:

```
# Configure item pipelines
# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'tutorial01.pipelines.Tutorial01Pipeline': 300,
}
```

分配给每个类的额整数值，确定了他们运行的顺序，item按数字从低到高的顺序，通过pipeline，通常将这些数字定义在0-1000范围内。

0-1000随意设置，数值越低，组建的优先级越高。

让我们重新执行爬取过程，尝试pipeline定义后的效果。

> scrapy crawl quotespider

可以看到文件中按json格式写出了每次解析得到的item。

将items写到MongoDB中

在下面的例子中，我们使用pymongo将items写到MongoDB中。MongoDB的地址和数据库名将在 scrapy 项目的 settings 中设置。例子主要显示了如何使用form_crawler()方法，以及如何正确的清空资源。

In [ ]:
```
"""一个用于将items存储到MongoDB的例子"""
import pymongo

class MongoPipeline(object):

    collection_name = 'scrapy_items'

    def __init__(self, mongo_uri, mongo_db):
        self.mongo_uri = mongo_uri
        self.mongo_db = mongo_db
    """form_crawler被定义为类方法，用于返回cls"""
    @classmethod
    def from_crawler(cls, crawler):
        return cls(
            mongo_uri=crawler.settings.get('MONGO_URI'),
            mongo_db=crawler.settings.get('MONGO_DATABASE', 'items')
        )

    def open_spider(self, spider):
        self.client = pymongo.MongoClient(self.mongo_uri)
        self.db = self.client[self.mongo_db]

    def close_spider(self, spider):
        self.client.close()

    def process_item(self, item, spider):
        self.db[self.collection_name].insert_one(dict(item))
        return item
```

对**items**进行截图

下面的例子显示了如何从process_item()延迟返回。例子中使用了Splash来获取item url的截图。

Pipeline生成了请求来本地运行Splash实例。在该请求被下载和延迟回调被激活时，它将把item保存为一个文件，并给每个item一个文件名。

重复值过滤器

下面的例子演示了如何使用对item的重复值进行过滤。重复值将被丢弃。

In [ ]:
```
from scrapy.exceptions import DropItem

class DuplicatesPipeline(object):

    def __init__(self):
        self.ids_seen = set()

    def process_item(self, item, spider):
        if item['id'] in self.ids_seen:
            raise DropItem("Duplicate item found: %s" % item)
        else:
            self.ids_seen.add(item['id'])
            return item
```

Logging

Scrapy提供了log功能，可以通过 logging 模块使用。

可以修改配置文件settings.py，任意位置添加下面两行，效果会清爽很多。

```
LOG_FILE = "TencentSpider.log"
LOG_LEVEL = "INFO"
```

Log levels

Scrapy提供5层logging级别:

- CRITICAL - 严重错误(critical)
- ERROR - 一般错误(regular errors)
- WARNING - 警告信息(warning messages)
- INFO - 一般信息(informational messages)
- DEBUG - 调试信息(debugging messages)

logging设置

通过在setting.py中进行以下设置可以被用来配置logging:

- LOG_ENABLED 默认: True，启用logging
- LOG_ENCODING 默认: 'utf-8'，logging使用的编码
- LOG_FILE 默认: None，在当前目录里创建logging输出文件的文件名
- LOG_LEVEL 默认: 'DEBUG'，log的最低级别
- LOG_STDOUT 默认: False 如果为 True，进程所有的标准输出(及错误)将会被重定向到log中。例如，执行 print "hello" ，其将会在Scrapy log中显示。
- LOG_FORMAT

尝试写日志

可以在spider文件中增加类似下列的语句:

```
import logging

logging.warning("This is a warning")
```

运行下列命令:

> scrapy crawl quotespider

可以看到在命令行控制台上显示有:

> 2018-10-30 15:26:19 [root] WARNING: This is a warning message.

如果需要指明由哪个文件写的日志，可以使用下列语句替代上面的代码:

```
import logging

logger = logging.getLogger('__name__')
logger.warning("This is a warning")
```

运行爬虫后会有如下显示:

> 2018-10-30 15:33:19 [tutorial01.spiders.quotespider] WARNING: This is a warning

如果要将log存入文件，可以使用命令行命令：

> scrapy crawl quotespider --logfile 1.log