法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：大数据分析挖掘
  - 新浪微博：ChinaHadoop

分布式爬虫

大纲

- Python Virtual Env
- Scrapy

Python Virtual Env

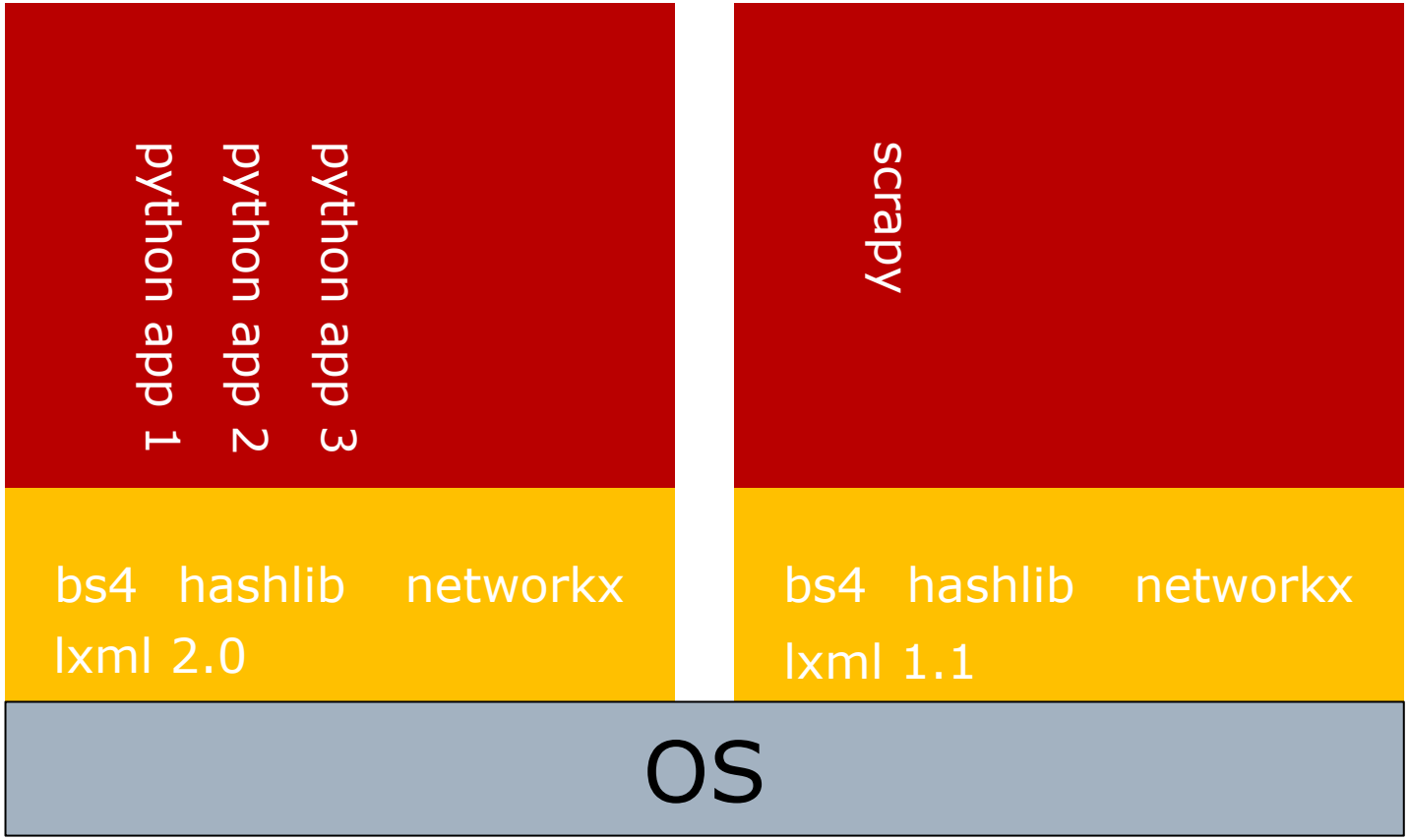
virtualenv – 基本介绍

官方建议是安装在 virtualenv 里

pip install virtualenv

Virtual Env 提供了一个完全独立、隔离的Python运行环境，简单说，在一个virtual env内部，python 运行环境拥有一个独立的文件夹，所有的python 环境都需要重新部署，python 代码存储和运行都是独立的

virtualenv – 用法



virtualenv – 基本语法

基本命令

virtualenv ENV 在当前目录会被新建一个名为ENV的文件夹，并把虚拟环境安装在 ENV 目录下

source bin/activate

把当前的路径加入到系统PATH的最前面，也就是会优先使用当前的路径下的可执行文件

退出：deactivate

virtualenv – 创建的参数

在创建的时候，可以使用以下参数：

--system-site-packages 会在安装的时候继承系统的package

--relocatable 使得虚拟环境今后可以迁移，当加上这个参数后，里面所有的路径都使用了相对路径，不建议使用

--extra-search-dir 讲自定义的路径加入到虚拟环境

Scrapy

爬虫框架概述

- 封装了网页下载、url 队列管理
- 深度管理
- 数据库支持
- 基本的存储管理
- 数据提取
- 封装（或集成了）元素选择器

Scrapy Core Features

- 模板化创建工程、启动爬虫的脚本
- 面向对象的编程模型
- 内容、数据抽取的各类接口
- 封装了 Xpath 及 CSS 选择器
- 一个 shell 的控制台 (Ipython aware），用来对下载的网页实验 CSS 及 Xpath 表达式以进行数据提取及debug
- 可以将数据转化为 JSON CSV XML 等多种格式，并支持 FTP/S3/Local FS 的存储
- 支持多种文本编码格式，并支持自动识别编码格式
- 强大的插件支持，可以自己挂载插件到 Scray 中来做功能增强
- Telnet Console 进入到 Python console 来监控和调试爬虫
- 自动的图片异步下载

Scrapy 内建的中间件

- cookie 和 session 处理
- HTTP 的 压缩、认证以及cache
- User-Agent 设置
- robots.txt 协议自动处理
- 爬取深度限制

Scrapy 核心 - Spider

Spider 是 Spiders 包下的一个类，是爬虫的入口和基本单元。一个 Scrapy 工程可以包含多个 Spider 类，每个 Spider 类定义了基本的抓取元素以及实现了抓取后的每个网页的处理方式，以及如何递归

最基本的，我们可以自定义或者利用 Scrapy 的命令来创建一个 Spider 的子类，然后通过命令 scrapy runspider xxx.py 或者 scrapy crawl xxx 来运行这个spider，一个爬虫任务就开始了

Scrapy project procedure

- Creating a new Scrapy project
- Writing a spider to crawl a site and extract data
- Exporting the scraped data using the command line
- Changing spider to recursively follow links

Create Scrapy Project

scrapy startproject tuortual

```
tutorial/
    scrapy.cfg            # deploy configuration file

    tutorial/             # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

Scrapy – Sample 1

```python
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/tag/humor/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.xpath('span/small/text()').extract_first(),
            }

        next_page = response.css('li.next a::attr("href")').extract_first()
        if next_page is not None:
            next_page = response.urljoin(next_page)
            yield scrapy.Request(next_page, callback=self.parse)
```

Sample – Sample 1 说明

- 运行：# scrapy runspider quotes_spider.py -o quotes.json
- 继承于 from scrapy.Spider
- start_urls：设置爬取的种子站点
- parse: callback 函数，当一个页面下载完成后后，调用parse来解 析和做数据抽取，这是一个可选的函数
- 用到了内建的 xpath 和 css 选择器
- urljoin(): 自动将相对网页build 成完整的网页

  'www.mafengwo.cn' + '/i/0122832.html'
- yield: python 用法，把结果加入到 generator 对象，该generator 对象在函数结束时候作为return对象返回
- Request(): 生成一个新的Request对象，加入到scheduler 并由 Downloader 下载

Sample – output


```
[
{
   "author": "Jane Austen",
   "text": "\u201cThe person, be it gentleman or lady.\u201d"},
{
   "author": "Groucho Marx",
   "text": "\u201cOutside of a dog, a book to read.\u201d"},
{
   "author": "Steve Martin",
   "text": "\u201cA day without sunshine like, night.\u201d"},
...
]
```

Scrapy – Sample 2

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotesample2"

    def start_requests(self):
        urls = [
            'http://quotes.toscrape.com/page/1/',
            'http://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = 'quotes-%s.html' % page
        with open(filename, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s' % filename)
```

Sample – Sample 2 说明

- 运行：# scrapy crawl quotesample2
- start_request: 继承自Spider，程序开始后由 Scrapy 调用，必须返回一个 Request 对象，与 Sample1 一样，Reqeust 对象会被加入到 scheduler 并实现异步下载
- python 的 with 语法：

```
with context_expression [as target(s)]:
        with-body
```

特定对象实现了“上下文管理协议”，例如 file, thread，他们实现了 __enter__() 和 __exit__() 方法，在 with 进入的时候 __enter__ 被调用，无论是否出现异常，with_body 在执行完成后都会调用 __exit__()来进行清理

优势：代码更简单，比如不需要加 try – catch 这样的语句

Sample – Sample 2 With

1. 执行 context_expression，生成上下文管理器 context_manager
2. 调用上下文管理器的 __enter__() 方法；如果使用了 as 子句，则将 __enter__() 方法的返回值赋值给 as 子句中的 target(s)
3. 执行语句体 with-body
4. 不管是否执行过程中是否发生了异常，执行上下文管理器的 __exit__() 方法，__exit__() 方法负责执行“清理”工作，如释放资源等。如果执行过程中没有出现异常，或者语句体中执行了语句 break/continue/return，则以 None 作为参数调用 __exit__(None, None, None) ；如果执行过程中出现异常，则使用 sys.exc_info 得到的异常信息为参数调用 __exit__(exc_type, exc_value, exc_traceback)
5. 出现异常时，如果 __exit__(type, value, traceback) 返回 False，则会重新抛出异常，让with 之外的语句逻辑来处理异常，这也是通用做法；如果返回 True，则忽略异常，不再对异常进行处理

Commands - genspider

- genspider [-t template] <name> <domain>
  - 在当前文件夹内，利用模板创建一个新的spider，或者在当前工程的spiders文件夹里（参考startporject创建的工程的文件夹架构）创建一个新的spider
  - 一个 spider 就是继承了 scrapy.Spider，并包含了 start_url 以及基本规则的 python 文件

template:

basic 几乎空的spider，只下载domain，不保存不解析

crawl 创建基于 scrapy.spiders.CrawlSpider 对象的模板，定义了如何提取link以及Items 的信息，详情参考 Items 及 样例 spider_rule

csvfeed 创建包含了可以按行解析的模板 pass_row()

xmlfeed 创建包含解析xml文件节点的模板 parse_node()

Commands 运行

- scracpy runspider <spider-file>

  运行一个已经存在的 spider
- scracpy crawl <spider-name>

  运行一个已经存在的 spider，通过 spider 名字来运行
- scracpy fetch url

  下载一个网页并在命令行里输出，用于简单实验

  --headers 打印HTTP的header

  --spider=SPIDER 用指定的爬虫来运行，指定的爬虫的参数会覆盖默认参数，例如 user-agent
- scrapy bench

  一个快速的 benchmark 测试，会显示下载流量、时间、响应数、内存消耗、response code 等

Commands 快速调试 - Shell

scrapy shell url

下载制定url后，打开shell脚本，允许你通过命令的方式来运行，类似于python shell，对于调试 xpath 及 css 非常有用，主要用法：

- 方法：sleep()、fetch(request)、view(response)，开始新的request、查看一个response等
- 变量（属性）： crawler、spider、request、response、settings，可以直接查看以上变量的值
- 在代码里注入：

```
from scrapy.shell import inspect_response
    inspect_response(response, self)
```

Commands快速调试- Parse

scrapy parse <url> [options]

快速下载制定网页，可以指定spider，然后调用parse函数来提取数据。可以指定 parse 方法以及rule。主要参数包括：

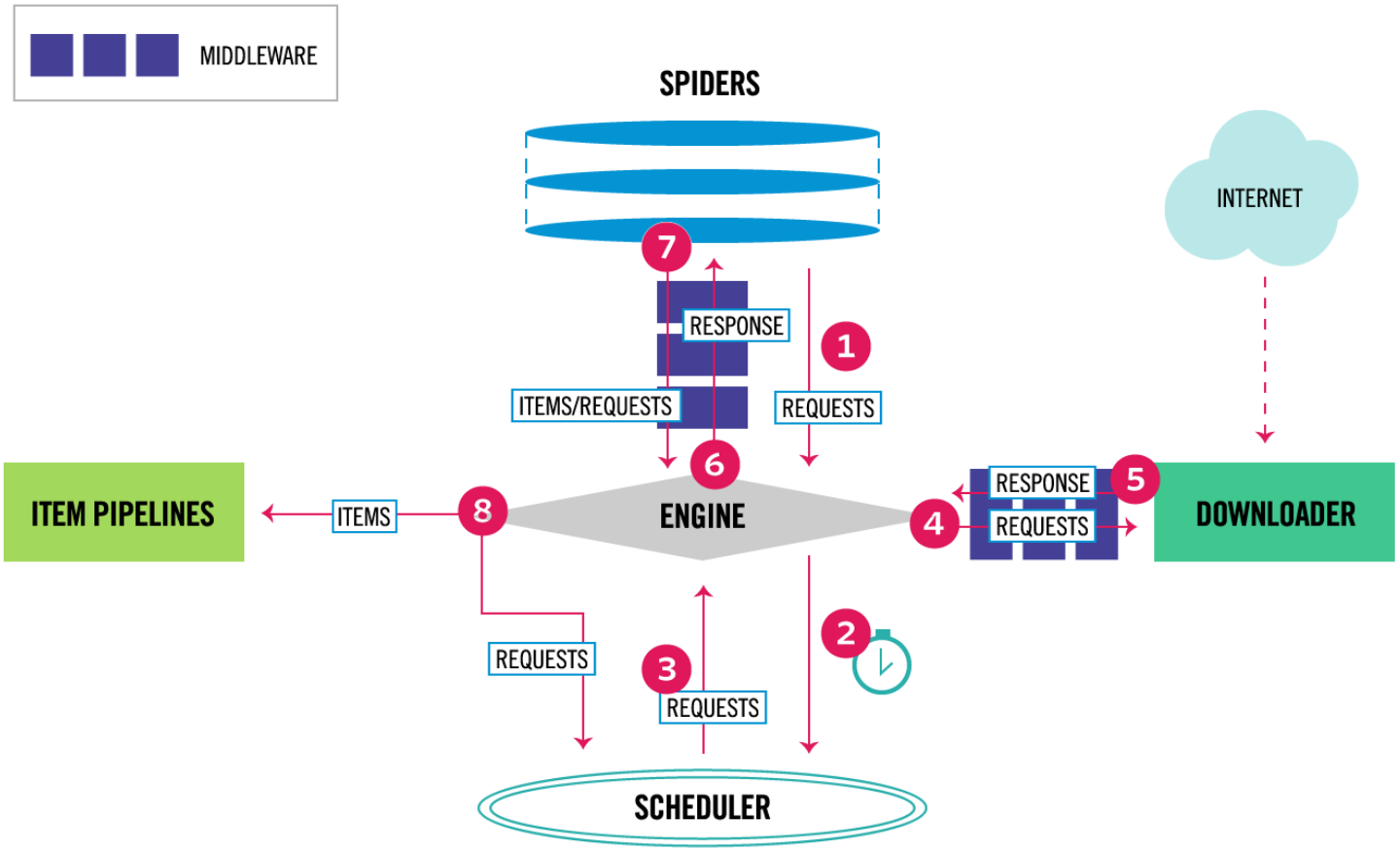
- --spider=SPIDER 用指定的SPIDER来抓取
- --callback or –c: 指定用于parse的方法

类似于shell，使用 parse 方法，可以快速测试所写的 spider 代码的下载链接、规则及解析函数

Spider work flow

1. start_request() start_urls: 指定开始抓取的网页
2. parse(): 网页下载完成后后，调用parse开始处理。parse可以同时返回解析后的数据，比如 dict，Items 对象，同时也可以返回 Request 对象。必须以 yield 方式返回一个可以遍历的对象。scrapy 会根据返回的 iterable 里的对象的类型来处理，例如取到的如果是 item，会作为解析对象放回结果里；如果是 Request，会放入 scheduler，进入下载队列
3. 对于解析出来的Items，可以通过 Item Pipeline 写入数据库，或者 Feed exports 以JSON、XML等方式写入文件

Scrapy overview



Spider类

1. start_request() start_urls: 指定开始抓取的网页
2. parse(): 网页下载完成后后，调用parse开始处理。parse可以同时返回解析后的数据，比如 dict，Items 对象，同时也可以返回 Request 对象。必须以 yield 方式返回一个可以遍历的对象。scrapy 会根据返回的 iterable 里的对象的类型来处理，例如取到的如果是 item，会作为解析对象放回结果里；如果是 Request，会放入 scheduler，进入下载队列
3. 对于解析出来的Items，可以通过 Item Pipeline 写入数据库，或者 Feed exports 以JSON、XML等方式写入文件

Spider Argument

在运行 crawl runspider 的时候通过 –a 可以给Spider传递参数，

例如:

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

```
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.e.com/categories/%s' % category]
     #...
```

CrawlSpider

scrapy.spiders.CrawlSpider

针对一些通用的网页抓取所设计的基础类，提供了比较简单的方式来处理外链以及定义一系列规则。通过 scrapy genspider –t crawl 创建的 Spider 会使用 CrawlSpider 的 base class

需要重写的方法及属性：

- rule:定一个了抓取的规则，是一个元组，应用rule的优先级是按照rule在Rules元组的顺序来的
- parse_start_url(response)：特定针对第一个网页的处理

CrawlSpider Rule Sample

```
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()').extract()
        item['description'] = response.xpath('//td[@id="item_description"]/text()').extract()
        return item
```

CrawlSpider Rule

在运行 crawl runspider 的时候通过 –a 可以给Spider传递参数，

例如:

scrapy crawl myspider -a category=electronics

```
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.e.com/categories/%s' % category]
     #...
```

XMLFeed

直接抓取并分析xml文件，可以用于sitemap

iterator: iternodes xml html 三种方式来调用 xpath 或 re 解析node
itertag：node 名称
parse_node: 解析回调

```
from scrapy.spiders import XMLFeedSpider

class MySpider(XMLFeedSpider):
    name = 'mfwsitemap'
    allowed_domains = ['mafengwo.cn']
    start_urls = ['http://www.mafengwo.cn/sitemapIndex.xml']
    iterator = 'html'
    itertag = 'sitemap' # the tag name of node

    def parse_node(self, response, node):
        # from scrapy.shell import inspect_response
        # inspect_response(response, self)

        item = {}
        item['loc'] = node.xpath('loc').extract()
        item['lastmod'] = node.xpath('lastmod').extract()
        return item
```

XMLFeed 的坑

问题描述

```
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<sitemap>
<loc>http://www.mafengwo.cn/article-0.xml</loc>
<lastmod>2017-02-22</lastmod>
</sitemap>
```

xmlns 声明了xml解析的namespace，所有根节点及子节点都将被放在 http://www.sitemaps.org/schemas/sitemap/0.9 的namespace 下，而 XPath 默认是查找没有 namespace 的，因此查找不满足条件

解决方案

使用 response.selector.remove_namespaces() 方法去除namespace ，然后再使用 xpath，例如 response.xpath("//sitemap")

Spider Argument

在运行 crawl runspider 的时候通过 –a 可以给Spider传递参数，

例如:

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

```
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.e.com/categories/%s' % category]
     #...
```

Spider Items

Items 本质上就是一个 python dictionary 的封装

```
import scrapy

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

Creating items

```
>>> product = Product(name='Desktop PC', price=1000)
>>> print product
Product(name='Desktop PC', price=1000)
```

Spider Items

Items 本质上就是一个 python dictionary 的封装

```
import scrapy

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

Creating items

```
>>> product = Product(name='Desktop PC', price=1000)
>>> print product
Product(name='Desktop PC', price=1000)
```

Spider Items Getter

Getting field values¶

```
>>> product['name']
Desktop PC
>>> product.get('name')
Desktop PC

>>> product['price']
1000
```

```
>>> product.keys()
['price', 'name']

>>> product.items()
[('price', 1000), ('name', 'Desktop PC')]
```

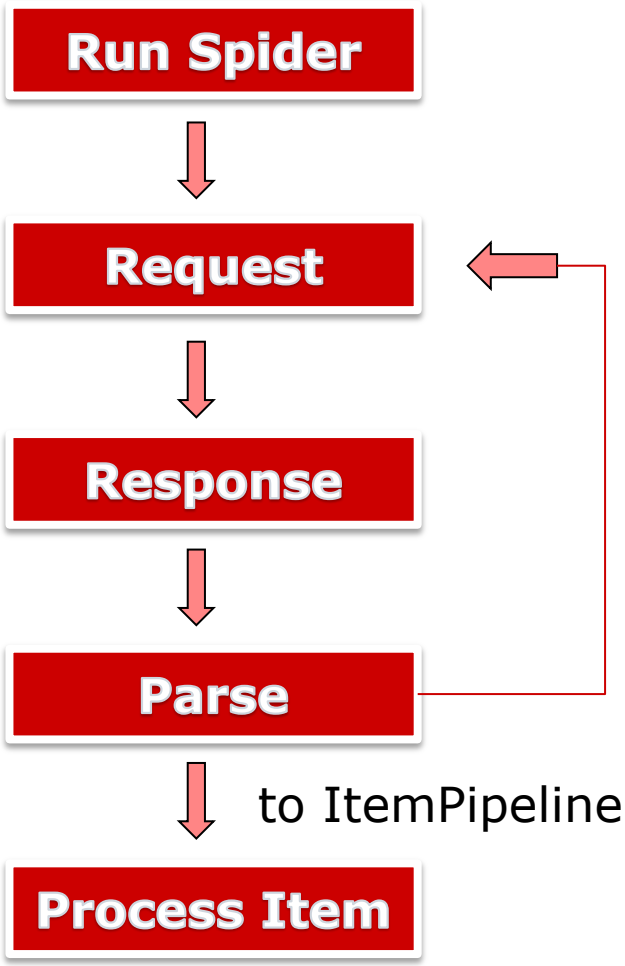
ItemLoader

Item 与 ItemLoader 配合使用，从response里提取数据并生成Item对象返回

```
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 values
    return l.load_item()
```

Item Pipleline



1. 创建 Item Pipeline 类，实现至少 process_item 这个方法

process_item(self, item, spider)

2. 将 pipeline 加入到 settings.py

```
# key is class of pipleline
# value is priority, lower value first

ITEM_PIPELINES = {
   'basicsample.pipelines.JDMongoPipelines':
300
}
```

File and Image Pipelines

文件及图片下载的 pipelines 支持：

- 避免重复下载
- 设置文件或图片的下载地址

对于图片，额外的支持：

- 下载的图片格式转换
- 生成缩略图
- 提取图片的高度及宽度

File Pipeline development

1. In a Spider, you scrape an item and put the URLs of the desired into a `file_urls` field.
2. The item is returned from the spider and goes to the item pipeline.
3. When the item reaches the **FilesPipeline**, the URLs in the `file_urls` field are scheduled for download using the standard Scrapy scheduler and downloader (which means the scheduler and downloader middlewares are reused), but with a higher priority, processing them before other pages are scraped. The item remains “locked” at that particular pipeline stage until the files have finish downloading (or fail for some reason).
4. When the files are downloaded, another field (`files`) will be populated with the results. This field will contain a list of dicts with information about the downloaded files, such as the downloaded path, the original scraped url (taken from the `file_urls` field) , and the file checksum. The files in the list of the `files` field will retain the same order of the original `file_urls` field. If some file failed downloading, an error will be logged and the file won’t be present in the `files` field.

File and Image Pipelines

Enabling your Media Pipeline

To enable your media pipeline you must first add it to your project **ITEM_PIPELINES** setting.

```
ITEM_PIPELINES = {'scrapy.pipelines.images.ImagesPipeline': 1}
```

```
ITEM_PIPELINES = {'scrapy.pipelines.files.FilesPipeline': 1}
```

Set the FILES_STORE or IMAGES_STORE setting:

```
FILES_STORE = '/path/to/valid/dir'
```

```
IMAGES_STORE = '/path/to/valid/dir'
```

File and Image Pipelines

Items 里定义 images_urls 和 images，这些field会被传入 pipelines 处理

```
import scrapy

class MyItem(scrapy.Item):

    # ... other item fields ...
    image_urls = scrapy.Field()
    images = scrapy.Field()

    # ... other item fields ...
    file_urls = scrapy.Field()
    files= scrapy.Field()
```

images_urls images 都是可以在 settings 里自定义，分别是

IMAGES_URLS_FIELD 及 IMAGES_RESULT_FIELD

完整做一个scrapy spider 任务

1. scrapy startproject chdemo 创建新的工程
2. scrapy genspider mfw mafengwo.com 创建mfw的spider
3. 编辑 Settings，设置 ITEM_PIPELINES 以及 IMAGES_STORE
4. scrapy shell http://www.mafengwo.cn/i/1082510.html 调试这个网页，重点测试 xpath 的选择
5. 编辑 spider/mfw.py 设置start_url
6. 修改parse文件
    1. yield iterable 对象，用于提取
    2. 提取外链 yield Request
7. scrapy crawl mfw

Scrapy Telnet

```
# telnet localhost 6023
```

Shortcut	Description
`crawler`	the Scrapy Crawler (**scrapy.crawler.Crawler** object)
`engine`	Crawler.engine attribute
`spider`	the active spider
`slot`	the engine slot
`extensions`	the Extension Manager (Crawler.extensions attribute)
`stats`	the Stats Collector (Crawler.stats attribute)
`settings`	the Scrapy settings object (Crawler.settings attribute)
`est`	print a report of the engine status
`prefs`	for memory debugging (see Debugging memory leaks)
`p`	a shortcut to the pprint.pprint function
`hpy`	for memory debugging (see Debugging memory leaks)

Scrapy Telnet

Telnet Samples

```
telnet localhost 6023
>> engine.pause() # pause the engine
>> engine.resume()
>> engine.stop()
>> est() # quickly show status of engine
Execution engine status

time()-engine.start_time                    : 8.62972998619
engine.has_capacity()                       : False
len(engine.downloader.active)               : 16
engine.scraper.is_idle()                    : False
```

配置telnet端口号（默认 6023 6073）

Settings.py

TELNETCONSOLE_PORT: [6023, 6073]

Middleware

中间件是可以挂载在Scrapy，用来处理Request、Response、Items 等

在设置里，添加自定义的中间件：

```
SPIDER_MIDDLEWARES = {
    'myproject.middlewares.CustomSpiderMiddleware': 543,
}
```

Scrapy 的很多功能也都是通过Middleware的方式实现的，例如下载是DownloadMiddleWare，它会检查 file_urls 并将其下载

Middleware

class scrapy.spidermiddlewares.SpiderMiddleware

process_spider_input(response, spider)

当有了response的时候被调用，必须返回 None（让Spider继续运行并交给其它中间件处理）或者抛出异常

process_spider_output(response, result, spider)

当Spider处理完成一个Response（即 parse 调用并得到了 iterable 后）调用这个方法，必须返回 iterable 的 Request， dict 或者 Item。其中参数 result 是spider已经处理完的结果，也就是 parse 返回的 Request dict 或 item

process_start_requests(start_requests, spider)

在 request 被发起的时候调用，与 process_spider_output 类似，只不过需要返回的是 Request 的队列

疑问

- 问题答疑： http://www.xxwenda.com/
  - 可邀请老师或者其他人回答问题

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：大数据分析挖掘
- 新浪微博：ChinaHadoop