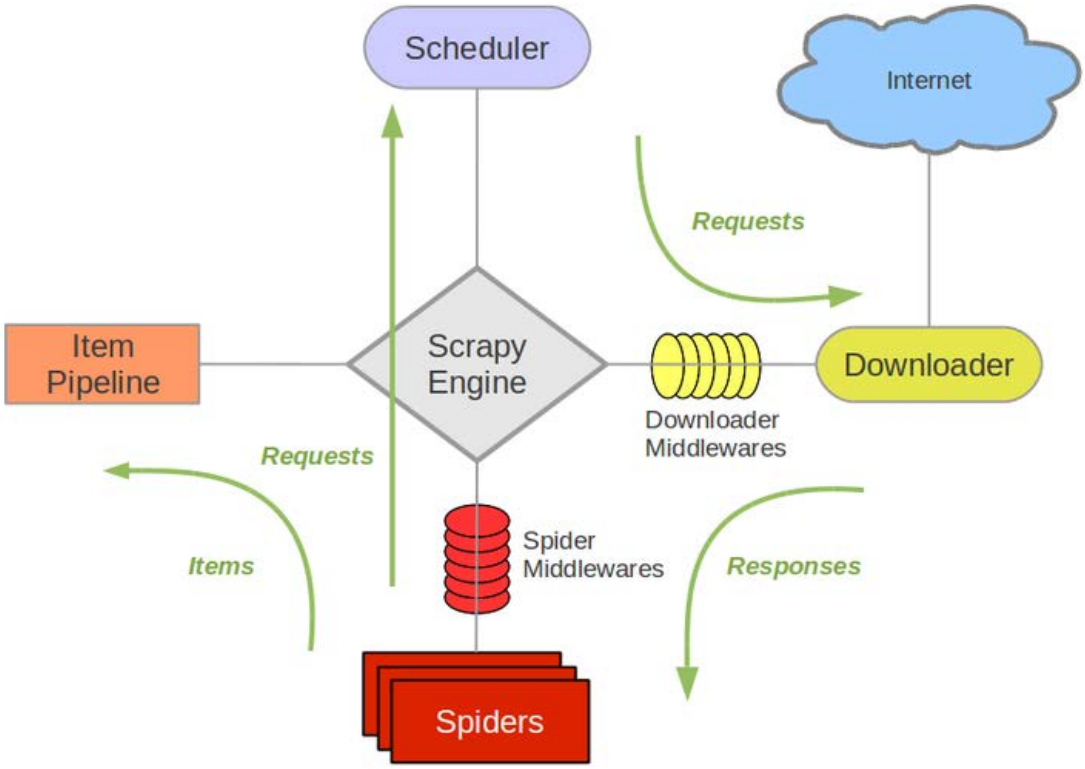
Scrapy 框架

- Scrapy 是用纯 Python 实现一个为了爬取网站数据、提取结构性数据而编写的应用框架。
- 用户只需要定制开发几个模块就可以轻松的实现一个爬虫，用来抓取网页内容以及各种图片。
- Scrapy 使用了 Twisted 异步网络框架来处理网络通讯，可以加快下载速度，并且包含了各种中间件接口，可以灵活的完成各种需求。

框架图



Scrapy Engine(引擎): 负责 Spider、ItemPipeline、Downloader、Scheduler 中间的通讯，信号、数据传递等。

Scheduler(调度器): 它负责接受引擎发送过来的 Request 请求，并按照一定的方式进行整理排列，入队，当引擎需要时，交还给引擎。

Downloader（下载器）: 负责下载 Scrapy Engine(引擎)发送的所有 Requests 请求，并将其获取到的 Responses 交还给 Scrapy Engine(引擎)，由引擎交给 Spider 来处理，

Spider（爬虫）: 它负责处理所有 Responses,从中分析提取数据，获取 Item 字段需要的数据，并将需要跟进的 URL 提交给引擎，再次进入 Scheduler(调度器)，

Item Pipeline(管道)：它负责处理 Spider 中获取到的 Item，并进行进行后期处理（详细分析、过滤、存储等）的地方.

Downloader Middlewares（下载中间件）: 一个可以自定义扩展下载功能的组件。

Spider Middlewares（Spider 中间件）: 可以扩展操作引擎和 Spider 中间通信的功能组件

安装配置

Windows 平台

pip install twisted
pip install lxml
pip install pywin32
pip install Scrapy

注意：一定安装 scrapy 的依赖库，否则可能会遇到诸多错误：

- Scrapy error: Microsoft Visual C++ 10.0 is required.
- Failed building wheel for lxml
- Failed building twisted
- Unable to find vcvarsall.bat

Ubuntu 平台

安装非 Python 的依赖

sudo apt-get install python-dev python-pip libxml2-dev libxslt1-dev zlib1g-dev libffi-dev libssl-dev

安装 Scrapy 框架

pip install --upgrade pip（选）
sudo pip install scrapy

测试

命令行输入 scrapy，提示类似以下结果，代表已经安装成功

```
(env1) C:\Users\admin>scrapy
Scrapy 1.5.0 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  runspider     Run a self-contained spider (without creating a proj
ect)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy

  [ more ]      More commands available when run from project direct
ory

Use "scrapy <command> -h" to see more info about a command
```

制作 Scrapy 爬虫步骤

- 新建项目（scrapy startproject xxx）：新建一个新的爬虫项目
- 明确目标 （编写 items.py）：明确你想要抓取的目标
- 制作爬虫 （spiders/xxspider.py）：制作爬虫开始爬取网页

- 存储内容 （pipelines.py）：设计管道存储爬取内容

入门案例

1、项目创建

scrapy startproject JobSpider

目录结构：

scrapy.cfg ： 项目的配置文件
jobSpider/ ： 项目的 Python 模块，将会从这里引用代码
jobSpider /items.py ： 项目的目标文件
jobSpider /pipelines.py ： 项目的管道文件
jobSpider /settings.py ： 项目的设置文件
jobSpider /spiders/ ： 存储爬虫代码目录

2、确定目标

抓取上海地区 python 招聘职位的情况，包括职位名称，工作地点，薪资，发布时间，url：

https://search.51job.com/list/020000,000000,0000,00,9,99,python,2,1.html?lang=c&stype=1&postchannel=0000&workyear=99&cotype=99&degreefrom=99&jobterm=99&companysize=99&lonlat=0%2C0&radius=-1&ord_field=0&confirmdate=9&fromType=&dibiaoid=0&address=&line=&specialarea=00&from=&welfare=

1. 打开 mySpider 目录下的 items.py

2. 创建一个 JobspiderItem 类，继承 scrapy.Item，构建 item 模型（model）。 并且

   定义类型为 scrapy.Field 的类属性，用来保存爬取到的数据

```
class JobspiderItem(scrapy.Item):
    # define the fields for your item here like:
    name = scrapy.Field()
    city = scrapy.Field()
    pub_date = scrapy.Field()
    salary = scrapy.Field()
```

3、制作爬虫

1、创建爬虫

scrapy genspider pythonPosition 51job.com

打开 mySpider/spider 目录里的 pythonPosition.py，默认增加了下列代码:

```
import scrapy

class PythonSpider(scrapy.Spider):
    name = "pythonPosition"
    allowed_domains = ["51job.com"]
    start_urls = (
        'http:// 51job.com/',
    )

    def parse(self, response):
        pass
```

说明

- name = ""：这个爬虫的识别名称，必须是唯一的，在不同的爬虫必须定义不同的名字。
- allow_domains = [] 是搜索的域名范围，也就是爬虫的约束区域，规定爬虫只爬取这个域名下的网页，不存在的 URL 会被忽略。
- start_urls = ()： 爬取的 URL 元祖/列表。爬虫从这里开始抓取数据，所以，第一次下载的数据将会从这些 urls 开始。其他子 URL 将会从这些起始 URL 中继承性生成。
- parse(self, response)： 解析的方法，每个初始 URL 完成下载后将被调用，调用的时候传入从每一个 URL 传回的 Response 对象来作为唯一参数，主要作用如下:
    - 负责解析返回的网页数据(response.body)，提取结构化数据(生成 item)
    - 生成需要下一页的 URL 请求。

将 start_urls 的值修改为需要爬取的第一个 url

```
start_urls =
("https://search.51job.com/list/020000,000000,0000,00,9,99,python,2,1.html?lang=c&stype=1&postchannel=0000&workyear=99&cotype=99&degreefro
```

m=99&jobterm=99&companysize=99&lonlat=0%2C0&radius=-1&ord_field=0&confirmdate=9&fromType=&dibiaoid=0&address=&line=&specialarea=00&from=&welfare=",)

2. 提取数据

F12 审查元素

重写 parse 方法，用 XPath 提取数据

```
def parse(self, response):
    #print(response.body)
    job_list = response.xpath("//div[@class='dw_table']/div[@class='el']")
    item = JobspiderItem()
    items = []
    for each in job_list:
        name = each.xpath("normalize-space(./p/span/a/text())").extract()[0]
        city = each.xpath("./span[@class='t3']/text()").extract()[0]
        pub_date = each.xpath(".//span[@class='t5']/text()").extract()[0]
        salary = each.xpath(".//span[@class='t4']/text()").extract()
        if len(salary)>0:
            salary = salary[0]
            salary = salary[:salary.index('/')]
        else:
            salary = ''
        print(name)
        print(city)
        print(pub_date)
        print((salary))
        item['name'] = name
        item['city'] = city
        item['pub_date'] = pub_date
        item['salary'] = salary
```

```
        # 将获取的数据交给 pipeline
        yield item
```

3、item 写入 csv 文件

```
import csv
import codecs

class JobspiderPipeline(object):
    def __init__(self):
        self.file = codecs.open('51job.csv', 'w', 'utf-8')
        self.wr = csv.writer(self.file, dialect="excel")
        self.wr.writerow(['name','pub_date','city','salary'])

    def process_item(self, item, spider):
        self.wr.writerow([item['name'], item['pub_date'], item['city'], item['salary']])
        return item

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

4、爬虫调试

1、创建 run.py 文件，和 setting.py 同级目录

2、添加代码：

```
from scrapy import cmdline
name = 'pythonPosition'
cmd = 'scrapy crawl {0}'.format(name)

cmdline.execute(cmd.split())
```

其中 name 参数为 spider 的 name。

3、接着在 spider 文件中设置断点。

4、run.py 文件中右键选择 Debug。

Spider

Spider 类定义了如何爬取某个(或某些)网站。包括了爬取的动作(例如:是否跟进链接)以及如何从网页的内容中提取结构化数据(爬取 item)。 换句话说，Spider 就是您定义爬取的动作及分析某个网页(或者是有些网页)的地方。

class scrapy.Spider 是最基本的类，所有编写的爬虫必须继承这个类。

主要用到的函数及调用顺序为:

__init__()：初始化爬虫名字和 start_urls 列表

start_requests() 调用 make_requests_from url():生成 Requests 对象交给 Scrapy 下载并返回 response

parse()：解析 response，并返回 Item 或 Requests (需指定回调函数)。Item 传给 Item pipline 持久化，而 Requests 交由 Scrapy 下载，并由指定的回调函数处理(默认 parse())，一直进行循环，直到处理完所有的数据为止。

yield 作用

参考实例：

```
def fab(max):
    n, a, b = 0, 0, 1
    while n < max:
        # print b
        yield b
        # print b
        a, b = b, a + b
        n = n + 1
print(fab(5))   # 输出：<generator object fab at 0x00000000069D8A68>
for n in fab(5):
     print n      # 依次 1,1,2,3,5
```

yield 解析：

yield 的作用就是把一个函数变成一个生成器(generator)，带有 yield 的函数不再是一个普通函数，Python 解释器会将其视为一个 generator，单独调用生成器函数,不会被执行而是返回一个 iterable 对象!

在 for 循环执行时，每次循环都会执行 fab 函数内部的代码，执行到 yield b 时，fab 函数就返回一个迭代值，

下次迭代时，代码从 yield b 的下一条语句继续执行，

而函数的本地变量看起来和上次中断执行前是完全一样的，于是函数继续执行，直到再次遇到 yield。

对 scrapy 中使用 yield 循环处理网页 url 的分析

def parse(self, response):

```
    # 具体处理逻辑：如，分析页面，找到页面中符合规则的内容（校花图片），保存
    hxs = HtmlXPathSelector(response)  # 创建查询对象
    # 获取所有的 url，继续访问，并在其中寻找相同的 url
    all_urls = hxs.select('//a/@href').extract()
    for url in all_urls:
        if url.startswith('http://www.xiaohuar.com/list-1-'):
            yield Request(url, callback=self.parse)  # 递归的找下去
            print(url)
```

python 将 parse()函数视为生成器

首先程序会将第一个 response 对象分析提取需要的东西，然后提取该 response 中所有的 urls 进行循环处理

首次执行到 parse-for-yield 处，会返回一个，即生成一个 Request1 对象并返回，这是一个迭代值，其中定义了回调方法为 parse

此时，第一次迭代结束。

第一次迭代过程中生成的 Request1 对象,即一个新的 url 请求,会返回一个新的 response,然后框架会使用该 response 执行回调函数，进行另一个分支的迭代处理

主要属性和方法

- **name**

  定义 spider 名字的字符串。

  例如，如果 spider 爬取 mywebsite.com ，该 spider 通常会被命名为 mywebsite

- **allowed_domains**

  包含了 spider 允许爬取的域名(domain)的列表，可选。

- **start_urls**

  初始 URL 元祖/列表。当没有制定特定的 URL 时，spider 将从该列表中开始进行爬取。

- **parse(self, response)**

  当请求 url 返回网页没有指定回调函数时，默认的 Request 对象回调函数。用来处理网页返回的 response，以及生成 Item 或者 Request 对象。

- **log(self, message[, level, component])**

  使用 scrapy.log.msg() 方法记录(log)message。

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:

- 验证爬取的数据(检查 item 包含某些字段，比如说 name 字段)
- 查重(并丢弃)
- 将爬取结果保存到文件或者数据库中

启用一个 Item Pipeline 组件

为了启用 Item Pipeline 组件，必须将它的类添加到 settings.py 文件 ITEM_PIPELINES 配置，比如:

```
ITEM_PIPELINES = {
    #'mySpider.pipelines.SomePipeline': 300,
    "mySpider.pipelines.JsonPipeline":300
}
```

分配给每个类的整型值，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，组件的优先级越高）

Request/ Response

Scrapy 使用 request 对象来爬取 web 站点。

request 对象

class scrapy.http.**Request**(url[, callback, method='GET', headers, body, cookies, meta, encoding='u

tf-8', priority=0, dont_filter=False, errback])

一个 request 对象代表一个 HTTP 请求，通常有 Spider 产生，经 Downloader 执行从而产生一个 Response。

参数:

url: 用于请求的 URL

callback:指定一个回调函数，该回调函数以这个 request 对应的 response 作为第一个参数。如果未指定 callback，则默认使用 spider 的 parse()方法。

method:HTTP 请求的方法，默认为 GET

meta:指定 Request.meta 属性的初始值，字典类型。可以在回调函数们之间传递参数

body:请求体

headers:request 的头信息。

cookies:cookie

encoding:请求的编码， 默认为 utf-8

priority:请求的优先级

dont_filter(boolean):指定该请求是否被 Scheduler 过滤。该参数可以是 request 重复使用（Scheduler 默认过滤重复请求）。谨慎使用!!

errback:处理异常的回调函数。

范例

```
def parse_page1(self, response):
    item = MyItem()
    item['main_url'] = response.url
    request = scrapy.Request("http://www.example.com/some_page.html",
                             callback=self.parse_page2)
    request.meta['item'] = item
    yield request
```

```
def parse_page2(self, response):
    item = response.meta['item']
    item['other_url'] = response.url
    yield item
```

Response 对象

一个 Response 对象表示的 HTTP 响应，这通常由下载器提供给到爬虫进行处理。

常见属性

url

包含响应的 URL 的字符串。

status

表示响应的 HTTP 状态的整数。示例：200， 404。

headers

包含响应标题的类字典对象。可以使用 get()返回具有指定名称的第一个标头值或 getlist()

返回具有指定名称的所有标头值来访问值。

例如，此调用会为您提供标题中的所有 Cookie：

response.headers.getlist('Set-Cookie')

body

正文

meta

获得 Request.meta 从您的爬虫发送的原始属性。

Selectors 选择器

Scrapy 提取数据有自己的一套机制。它们被称作选择器(seletors),

response.selector：获取到一个 response 初始化的类 Selector 的对象，此时可以通过使用 response.selector.xpath()或 response.selector.css() 来对 response 进行查询。

Selector 有四个基本的方法，最常用的还是 xpath:

xpath(): 传入 xpath 表达式，返回该表达式所对应的所有节点的 selector list 列表

extract(): 序列化该节点为 Unicode 字符串并返回 list

css(): 传入 CSS 表达式，返回该表达式所对应的所有节点的 selector list 列表，语法同 BeautifulSoup4

re(): 根据传入的正则表达式对数据进行提取，返回 Unicode 字符串 list 列表

快捷方式

response.xpath()

response.css()

XPath 表达式的例子及对应的含义:

/html/head/title: 选择<HTML>文档中 <head> 标签内的 <title> 元素

/html/head/title/text(): 选择上面提到的 <title> 元素的文字

//td: 选择所有的 <td> 元素

//div[@class="mine"]: 选择所有具有 class="mine" 属性的 div 元素