

Scrapy爬虫基本使用



The Website is the API ...

Requests

自动爬取HTML页面
自动网络请求提交

robots.txt

网络爬虫排除标准

**Beautiful
Soup**

解析HTML页面

Re

正则表达式详解
提取页面关键信息

Scrapy*

*网络爬虫原理介绍
*专业爬虫框架介绍

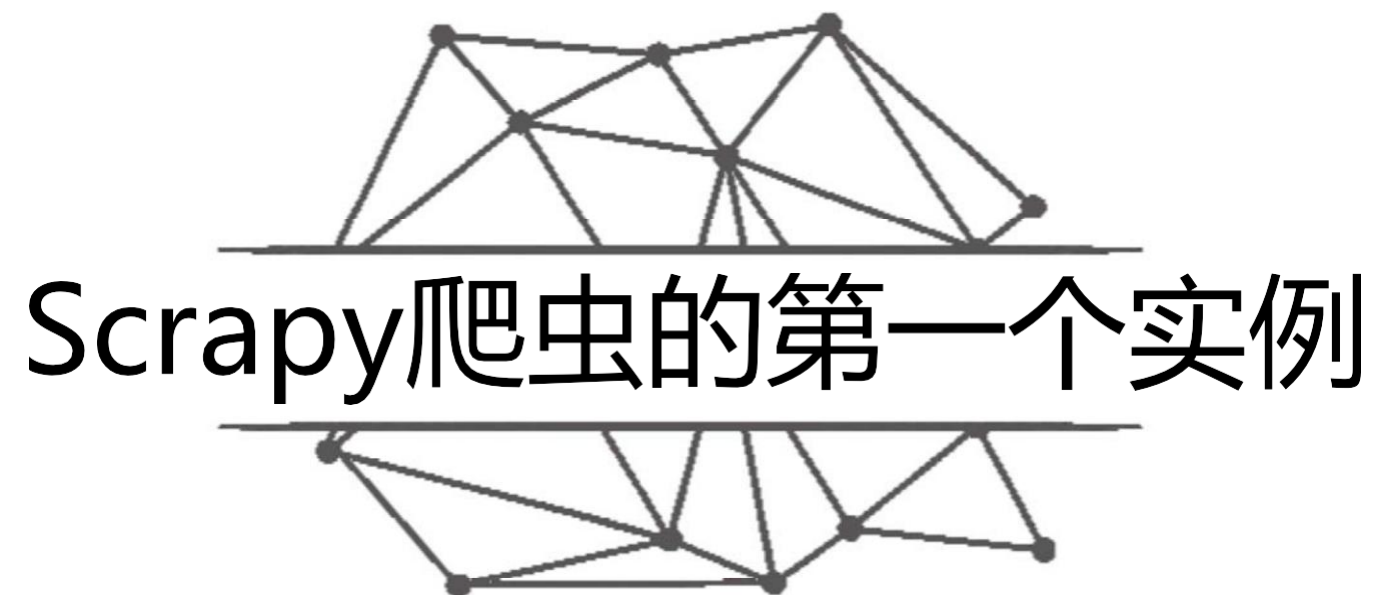
Projects

实战项目A/B

掌握定向网络数据爬取和网页解析的基本能力

python
弹指之间 · 享受创新

Python网络爬虫与信息提取

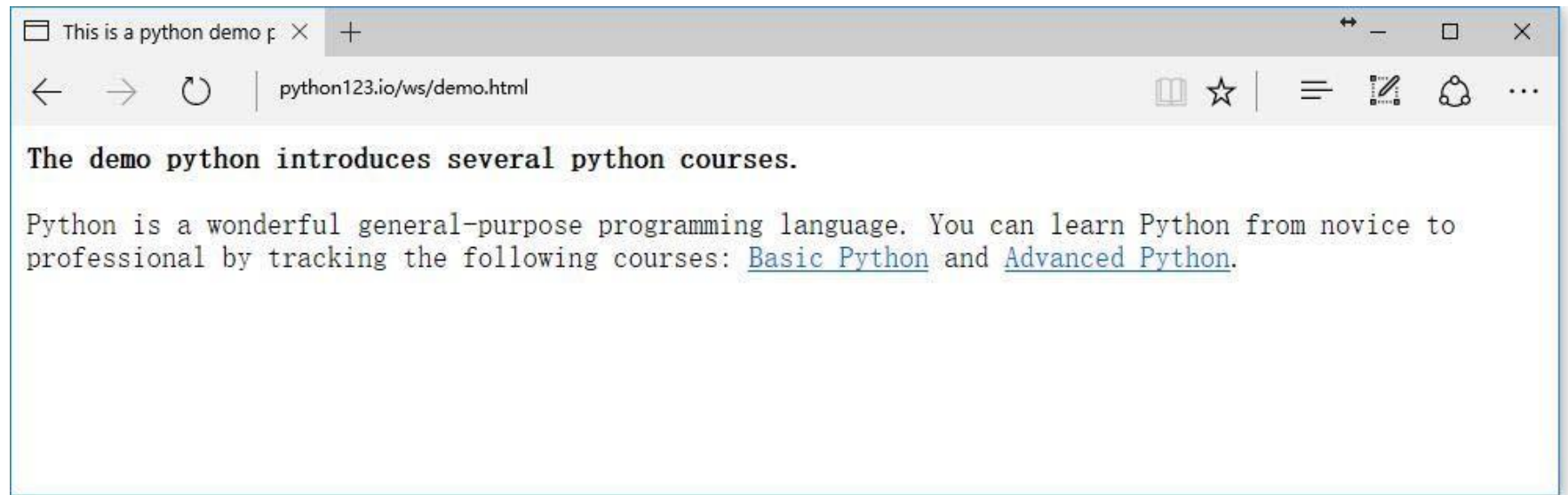


Scrapy爬虫的第一个实例

演示HTML地址

演示HTML页面地址：<http://python123.io/ws/demo.html>

文件名称：demo.html



产生步骤（1）

应用Scrapy爬虫框架主要是编写配置型代码

步骤1：建立一个Scrapy爬虫工程

选取一个目录（D:\pycodes\），然后执行如下命令：

```
D:\pycodes>scrapy startproject python123demo
```

生成的工程目录

目录结构	python123demo/	————→	外层目录
	scrapy.cfg	————→	部署 Scrapy爬虫的配置文件
	python123demo/	————→	Scrapy框架的用户自定义Python代码
	__init__.py	————→	初始化脚本
	items.py	————→	Items 代码模板（继承类）
	middlewares.py	————→	Middlewares代码模板（继承类）
	pipelines.py	————→	Pipelines代码模板（继承类）
	settings.py	————→	Scrapy爬虫的配置文件
	spiders/	————→	Spiders代码模板目录（继承类）
	__pycache__/	————→	缓存目录，无需修改

生成的工程目录

spiders/	————→	Spiders代码模板目录（继承类）
__init__.py	————→	初始文件，无需修改
__pycache__/	————→	缓存目录，无需修改

内层目录结构

用户自定义的spider代码增加在此处

产生步骤（2）

步骤2：在工程中产生一个Scrapy爬虫

进入工程目录（D:\pycodes\python123demo），然后执行如下命令：

```
D:\pycodes\python123demo>scrapy genspider demo python123.io
```

该命令作用：

- (1) 生成一个名称为demo的spider
- (2) 在spiders目录下增加代码文件demo.py

该命令仅用于生成demo.py，该文件也可以手工生成

demo.py文件

```
# -*- coding: utf-8 -*-
```

```
import scrapy
```

```
class DemoSpider(scrapy.Spider):
```

```
    name = "demo"
```

```
    allowed_domains = ["python123.io"]
```

```
    start_urls = ['http://python123.io/']
```

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

```
        pass
```

parse()用于处理响应，解析内容形成字典，
发现新的URL爬取请求

产生步骤 (3)

步骤3：配置产生的spider爬虫

配置：(1) 初始URL地址 (2) 获取页面后的解析方式

```
# -*- coding: utf-8 -*-
import scrapy

class DemoSpider(scrapy.Spider):
    name = "demo"
    # allowed_domains = ["python123.io"] #可选
    start_urls = ['http://python123.io/ws/demo.html']

    def parse(self, response):
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s.' % fname)
```

产生步骤（4）

步骤4：运行爬虫，获取网页

在命令行下，执行如下命令：

```
D:\pycodes\python123demo>scrapy crawl demo
```

demo爬虫被执行，捕获页面存储在demo.html

回顾demo.py代码

```
import scrapy

class DemoSpider(scrapy.Spider):
    name = "demo"
    start_urls = ['http://python123.io/ws/demo.html']

    def parse(self, response):
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s.' % fname)
```

demo.py代码的完整版本

```
import scrapy

class DemoSpider(scrapy.Spider):
    name = "demo"

    def start_requests(self):
        urls = [
            'http://python123.io/ws/demo.html'
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

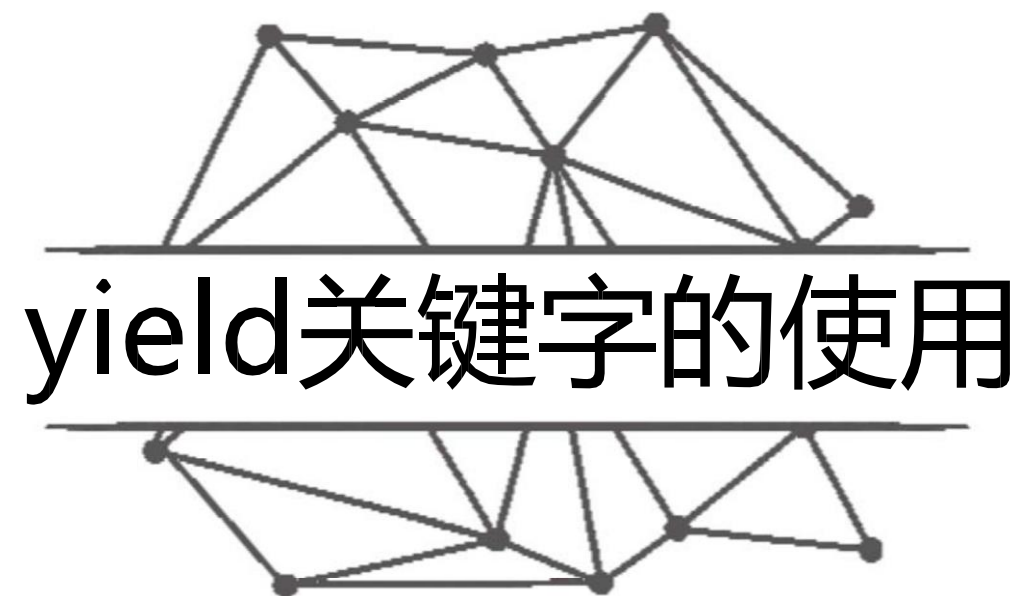
    def parse(self, response):
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s.' % fname)
```

demo.py两个等价版本的区别

```
class DemoSpider(scrapy.Spider):  
    name = "demo"  
    start_urls = ['http://python123.io/ws/demo.html']
```



```
class DemoSpider(scrapy.Spider):  
    name = "demo"  
  
    def start_requests(self):  
        urls = [  
            'http://python123.io/ws/demo.html'  
        ]  
        for url in urls:  
            yield scrapy.Request(url=url, callback=self.parse)
```



yield关键字

yield  生成器

包含yield语句的函数是一个生成器

生成器每次产生一个值（yield语句），函数被冻结，被唤醒后再产生一个值

生成器是一个不断产生值的函数

实例

```
>>> def gen(n):  
    for i in range(n):  
        yield i**2  
  
>>> for i in gen(5):  
    print(i, " ", end="")  
  
0  1  4  9  16  
>>>
```

生成器每调用一次在yield位置产生一个值，直到函数执行结束

为何要有生成器？

实例：求一组数的平方值

```
>>> def gen(n):  
    for i in range(n):  
        yield i**2
```

```
>>> for i in gen(5):  
    print(i, " ", end="")
```

```
0  1  4  9 16  
>>>
```

生成器写法

```
>>> def square(n):  
    ls = [i**2 for i in range(n)]  
    return ls
```

```
>>> for i in square(5):  
    print(i, " ", end="")
```

```
0  1  4  9 16  
>>>
```

普通写法

为何要有生成器？

生成器相比一次列出所有内容的优势：

1) 更节省存储空间

2) 响应更迅速

3) 使用更灵活

```
>>> def square(n):  
    ls = [i**2 for i in range(n)]  
    return ls
```

```
>>> def gen(n):  
    for i in range(n):  
        yield i**2
```

如果n=1M、10M、100M或更大呢？

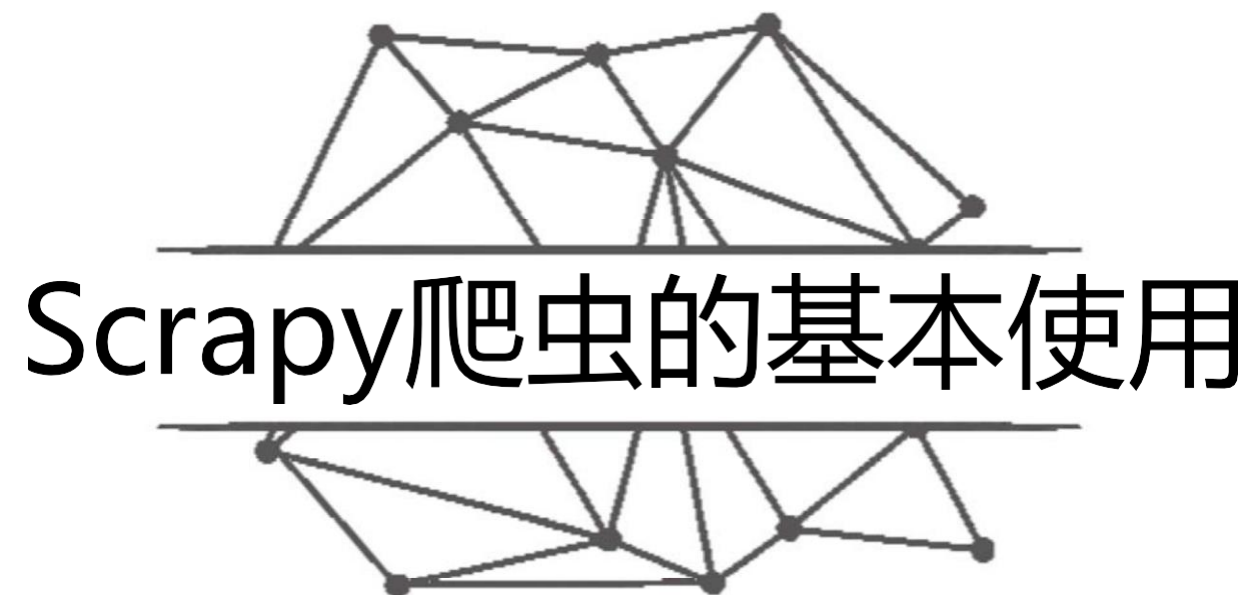
demo.py

```
import scrapy

class DemoSpider(scrapy.Spider):
    name = "demo"

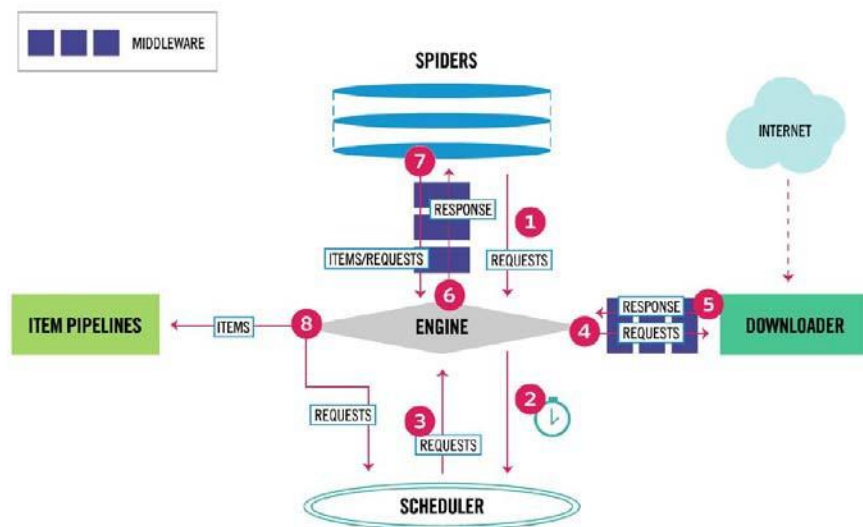
    def start_requests(self):
        urls = [
            'http://python123.io/ws/demo.html'
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s.' % fname)
```



Scrapy爬虫的基本使用

Scrapy爬虫的使用步骤



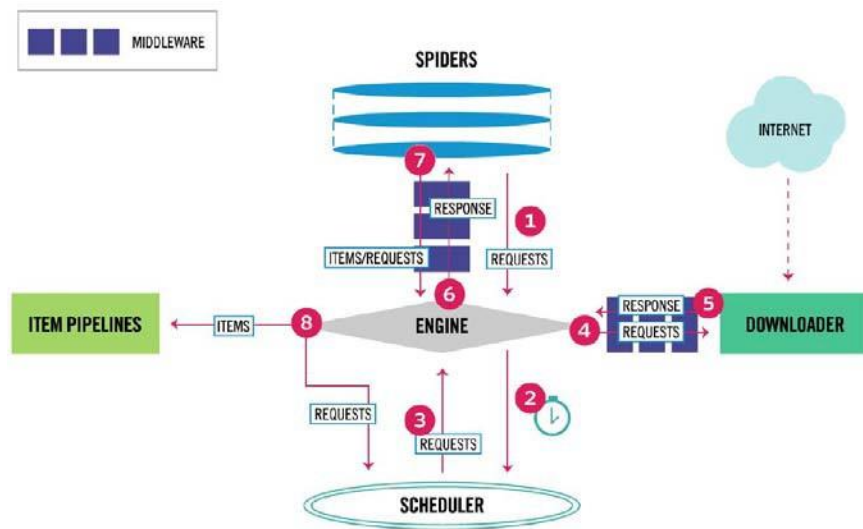
步骤1：创建一个工程和Spider模板

步骤2：编写Spider

步骤3：编写Item Pipeline

步骤4：优化配置策略

Scrapy爬虫的数据类型



Request类

Response类

Item类

Request类

```
class scrapy.http.Request()
```

Request对象表示一个HTTP请求
由Spider生成，由Downloader执行

Request类

属性或方法	说明
.url	Request对应的请求URL地址
.method	对应的请求方法，'GET' 'POST'等
.headers	字典类型风格的请求头
.body	请求内容主体，字符串类型
.meta	用户添加的扩展信息，在Scrapy内部模块间传递信息使用
.copy()	复制该请求

Response类

```
class scrapy.http.Response()
```

Response对象表示一个HTTP响应
由Downloader生成，由Spider处理

Response类型

属性或方法	说明
.url	Response对应的URL地址
.status	HTTP状态码，默认是200
.headers	Response对应的头部信息
.body	Response对应的内容信息，字符串类型
.flags	一组标记
.request	产生Response类型对应的Request对象
.copy()	复制该响应

Item类

```
class scrapy.item.Item()
```

Item对象表示一个从HTML页面中提取的信息内容

由Spider生成，由Item Pipeline处理

Item类似字典类型，可以按照字典类型操作

Scrapy爬虫提取信息的方法

Scrapy爬虫支持多种HTML信息提取方法：

- BeautifulSoup Soup
- lxml
- re
- XPath Selector
- CSS Selector

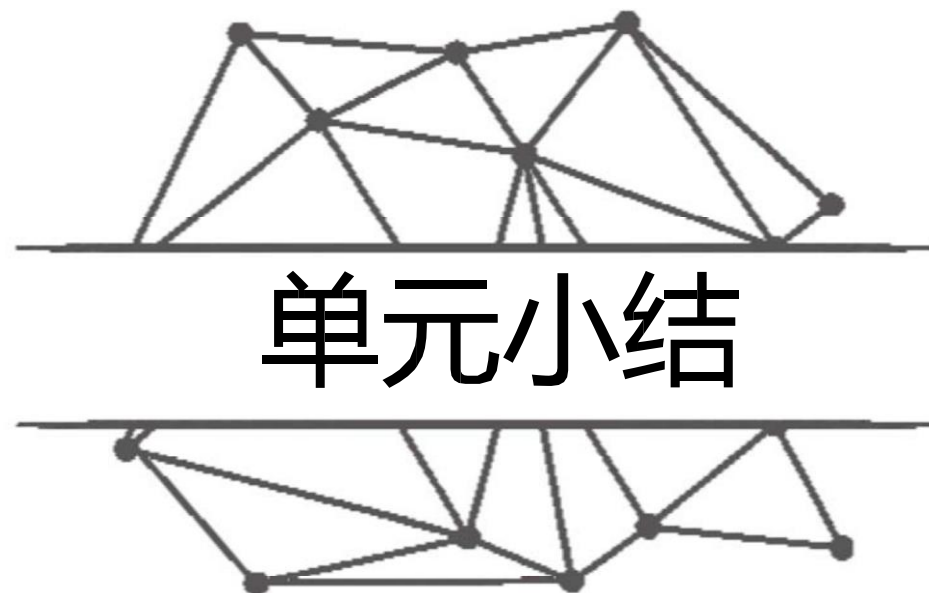
CSS Selector的基本使用

```
<HTML>.css('a::attr(href)').extract()
```

标签名称

标签属性

CSS Selector由W3C组织维护并规范



Scrapy爬虫基本使用

Scrapy爬虫的第一个例子及目录结构

yield关键字和生成器

Request

Response

Item

CSS Selector的基本使用